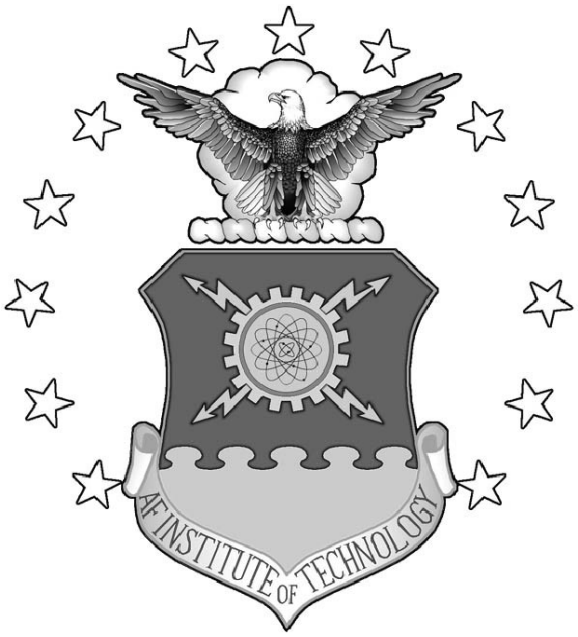

DIFFERENT FORMULATIONS OF THE ORTHOGONAL ARRAY PROBLEM AND THEIR SYMMETRIES

DISSERTATION

Andrew J. Geyer, Major, USAF

AFIT-ENC-DS-14-J-16

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

AFIT-ENC-DS-14-J-16

DIFFERENT FORMULATIONS OF THE ORTHOGONAL ARRAY PROBLEM AND THEIR SYMMETRIES

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy in Applied Mathematics (Statistics)

Andrew J. Geyer, M.S.

Major, USAF

June 2014

AFIT-ENC-DS-14-J-16

DIFFERENT FORMULATIONS OF THE ORTHOGONAL ARRAY PROBLEM AND THEIR SYMMETRIES

Andrew J. Geyer, M.S.
Major, USAF

Approved:

//signed//	27 May 2014
Dr. Dursun A. Bulutoglu, PhD (Chairman)	Date
//signed//	27 May 2014
Dr. Matthew C. Fickus, PhD (Member)	Date
//signed//	22 May 2014
Dr. Raymond R. Hill, PhD (Member)	Date

Accepted:

//signed//	27 May 2014
Dr. Adedeji B. Badiru Dean, Graduate School of Engineering and Management	Date

AFIT-ENC-DS-14-J-16

Abstract

Modern statistical experiments routinely feature a large number of input variables that can each be set to a variety of different levels. In these experiments, output response changes as a result of changes in the individual factor level settings. Often, an individual experimental run can be costly in time, money or both. Therefore, experimenters generally want to gain the desired information on factor effects from the smallest possible number of experimental runs. Orthogonal arrays provide the most desirable designs. However, finding orthogonal arrays is a very challenging problem.

There are numerous integer linear programming formulations (ILP) in the literature whose solutions are orthogonal arrays. Because of the nature of orthogonal arrays, these ILP formulations contain symmetries where some portion of the variables in the formulation can be swapped without changing the ILP. These symmetries make it possible to eliminate large numbers of infeasible or equivalent solutions quickly, thereby greatly reducing the time required to find all non-equivalent solutions to the ILPs.

In this dissertation, a new method for identifying symmetries is developed and tested using several existing and new ILP formulations for enumerating orthogonal arrays.

For my Dad, who always taught me the value of learning.

Acknowledgments

I would like to sincerely thank my advisor, Dr. Dursun Bulutoglu, for his countless hours of hard work and dedication. He went above and beyond at every turn to ensure the success of this research effort.

This research was supported by the AFOSR grant F1ATA03039J001.

Andrew J. Geyer

Table of Contents

Page

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
BILP	Binary Integer Linear Program
GMA	Generalized Minimum Aberration
GWP	Generalized Wordlength Pattern
LP	Linear Program
MILPs	Mixed Integer Linear Programs
MOLS	Mutually Orthogonal Latin Squares
OA	Orthogonal Array
OD	Orthogonal Design
OLS	Orthogonal Latin Squares
USAF	United States Air Force

DIFFERENT FORMULATIONS OF THE ORTHOGONAL ARRAY PROBLEM AND THEIR SYMMETRIES

I. Introduction

1.1 Motivation

Since the discovery of the scientific method in the 17th century, the analysis of experimental results has been the mechanism through which virtually all hypotheses have been tested. Montgomery [26] defines an *experiment* as "a test or series of runs in which purposeful changes are made to the input variables of a process or system so that we may observe and identify the reasons for changes that may be observed in the output response." Here in the early 21st century, human knowledge and technology have advanced so far that today's experiments routinely feature a large number of input variables (commonly called *factors*) that can each be set to several different *levels*. In these experiments, the output response changes as a result of changes in the individual factor level settings. One can easily see that for most modern experiments, conducting a *full factorial design* experiment, where all possible combinations of all factor levels are tested, can be too difficult to conduct either due to cost or time constraints. One prominent way to obtain the required output responses from an experiment at a lower cost than a full factorial experiment is to design an experiment, such as a fractional factorial design [26]. A *fractional factorial design* is essentially a subset of a full factorial design experiment where only a fraction of the runs are used. The fraction is selected to make optimal use of the sparsity-of-effects principle to isolate out the main effects and interactions of interest [26].

Fractional factorial designs have been used for product development and testing in industry for decades with great success. In particular, these designs have been used by

the United States Department of Defense to both improve the quality of weapons systems and other equipment as well as to lower maintenance, operation and replacement costs. The United States Air Force (USAF) is particularly dedicated to this type of testing and evaluation. The USAF operates a number of test and evaluation centers that examine and evaluate the full spectrum of its systems and equipment. These centers use designed experiments to test aircraft, motor vehicles, communication networks, avionics, radars, lasers, computer systems, automated information systems, directed energy weapons, ordinance, missiles, air traffic control systems, aircraft landing systems, reconnaissance platforms, nuclear weapons systems, weather sensors and space systems to name a few. These tests provide engineers with the data needed to make valid inferences about the capabilities of equipment and systems in order to improve quality and performance. More in-depth discussions of these types of tests conducted by the USAF test and evaluation community can be found in Tucker *et al.* [33], Hill and Chambal [14] and Hutto and Higdon [16]. It is clear from the level of effort put into testing and evaluation by the USAF that there is a need for efficient large-scale factorial designs to ensure that increasingly complex USAF systems operate at their peak of efficiency for the lowest possible cost.

When looking at costs, the USAF is not just concerned with evaluating the life cycle cost of its systems and equipment. The cost of testing and evaluation is also of great concern, particularly in times of shrinking federal budgets. As previously discussed, conducting test runs of factor level combinations can be very expensive and often a full factorial design experiment is prohibitively expensive. For example, the test of a particular missile system may cost several million dollars per missile test. In these cases, fractional factorial designs are best used to determine optimal operational factor settings. A properly selected fractional factorial design is critical if one wishes to obtain as much pertinent

information as possible at a fixed cost. It is well known that orthogonal arrays can serve as the paragon of efficient fractional factorial designs.

An Orthogonal Array (OA) with strength t is an array consisting of elements drawn from a fixed, finite set of symbols that are arranged in such a way that for every t columns of the array, all ordered t-tuples of the symbols appear an equal number of times. In statistical experiments, a factorial design $\mathbf{D}$ with N runs, k factors, and s levels is an OA of strength t, where $1 \leq t \leq k$, if each of the s^t t-level combinations appear exactly $\lambda = N/s^t$ times when $\mathbf{D}$ is projected onto any t factors. The integer λ is called the *index* of the OA. Such an OA is usually referred to using the shorthand OA(N, k, s, t). Xu and Wu [35] showed that, under the hierarchical ordering principle of factorial effects, a factorial design with a Generalized Wordlength Pattern (GWP) that is lexicographically smaller is more desirable. Hence, OAs with the Generalized Minimum Aberration (GMA) (designs that sequentially minimize the GWP criterion) are the most desirable designs. Finding such orthogonal arrays remains a very challenging problem.

OAs have additional applications outside the realm of designed experiments. In the 2012 U.S. Defense Strategic Guidance, the Secretary of Defense listed the ability to "operate effectively in cyberspace and space" as one of the primary missions of the U.S. Armed Forces [28]. In that same document, the Secretary further emphasized the need to build on advancements in network warfare and capitalize on the interconnected nature of every individual and piece of equipment on the modern battlefield [28]. As such, it is obvious that U.S. military information networks will only continue to grow in the future. In an unpredictable and fast-moving combat environment, network reliability will become essential to mission effectiveness. OAs can be used to assist in maintaining the security of these vital networks. In particular, Fecko and Steinder [10] demonstrated that 3-level OAs can be used for multiple fault localization in an unreliable environment like a modern battlefield. In their research, OAs were able to reduce the exponential number

of failure configurations in a battlefield network to a small enough subset that statistical analysis could be performed to locate the likely causes of failures. The ability to efficiently and accurately generate OAs would be of great use to a wide variety of research and testing organizations within the USAF as well as to the larger scientific and engineering community.

1.2 Research Contribution

The next chapter provides a review of recent best efforts to develop algorithms that generate GMA OA(N, k, s, t) for many values of N, k, s and t. These algorithms generate OAs by enumerating all solutions of different ILP formulations. The research in this dissertation seeks to build upon the current ILP-based algorithms for finding GMA or near-GMA OAs in Bulutoglu and Margot [7], Bulutoglu and Ryan [8] as well as new formulations developed from results in Rosenberg [29], Williamson [34] and Seberry and Yamada [31]. It features a thorough investigation of proposed enhancements to the best known algorithms for finding GMA OA(N, k, s, t). These enhancements are based on a generalization of the symmetry group of a Linear Program (LP) in Margot [23].

In Chapter 3, it is shown that there are no redundant inequalities in the LP relaxation of the Rosenberg [29] ILP that describes OA(N, k, s, t)s. This implies that each inequality is a facet of the LP relaxation. In Section 3.3, this result is used to determine the order of the symmetry group of the LP relaxation of the Rosenberg [29] ILP. The practical value of this result is that it exhibits the shortcomings of the Rosenberg [29] ILP LP relaxation symmetry group in describing the symmetries of the OA problem.

Bulutoglu and Margot [7] previously proved that the isomorphism group, the group that sends OAs to OAs by permuting factors or levels within each factor, is a subgroup of the ILP automorphism group that they used to enumerate OAs. Based on computational observations, they conjectured that these two groups are equal. This conjecture is proved in Chapter 3.

In Section 4.2.1, for many N, k, s, t combinations, the Margot ILP solver [22] is used to find a set of all non-isomorphic solutions to the Rosenberg [29] ILP by exploiting the symmetry group of its LP relaxation. For the same N, k, s, t combinations, this is compared to finding all non-isomorphic solutions to the Bulutoglu and Margot [7] ILP formulation by exploiting the larger isomorphism group. In most cases, enumerating a set of all non-isomorphic solutions to this ILP is faster, even though it has more variables. This provides a plethora of ILPs where exploiting a larger symmetry group more than overcomes the additional burden of having a larger number of variables. This is significant since it is contrary to the preprocessing step in mathematical programming that calls for deleting as many variables as possible from an ILP formulation.

It is essential to find larger subgroups of the ILP's symmetry group to speed up the Margot ILP solver [22] enumeration. The Margot ILP solver [22] uses the automorphism group of the formulation. This group is eqaul to the symmetry group of the LP relaxation, if the LP relaxation has no redundant inequalities and is full dimensional. Otherwise, a non-full dimensional LP relaxation may contain hidden symmetries not captured by the formulation automorphism group.

Section 4.2 results suggest that converting an ILP with inequality constraints into an equality constrained ILP by adding slack integer variables can be beneficial if one is to use the Margot ILP solver [22]. This, in turn, underscores the importance of finding larger subgroups of the symmetry group of an ILP with equality constraints. Currently, the Margot ILP solver [22] exploits the automorphism group of the ILP formulation. This group is always a subgroup of the LP relaxation symmetry group. Thus, it is more desireable to exploit the LP relaxation symmetry group. The LP relaxation of an ILP with equality constraints is not full dimensional. Hence, there is a need for algorithms that find the symmetry group of a general linear program with equality constraints. In Sections 4.2 and 4.3, an algorithm is developed that satisfies this need. This is the first time

in the literature that an algorithm for finding the symmetry group of a non-full dimensional linear program is developed.

In Section 4.2, the developed algorithm is implemented on LP relaxations of ILPs with equality constraints that describe orthogonal arrays. Computational results reveal previously unknown hidden symmetries of the Bulutoglu and Ryan [8] ILP formulation of the OA problem. Consequently, exploiting the newly found symmetries decreases solution times significantly.

In Section 4.3, the newly developed algorithm is tested with the LP relaxation symmetry groups of modified versions of Mixed Integer Linear Programs (MILPs) featured in Liberti [18]. It reveals hidden symmetries of the modified MILPs that could not otherwise be detected.

In general, the symmetry group of an ILP contains the symmetry group of its LP relaxation. One fundamental question is: "How close is the LP relaxation symmetry group to capturing all the symmetries in the ILP?" In Section 4.4, attempts are made to answer this question for the OA problem. This is accomplished by calculating the symmetry group of the Bulutoglu and Ryan [8] ILP, for several OA(N, k, s, t).

In Section 4.5, attempts were made to speed up the Bulutoglu and Ryan [8] OA extension algorithm by exploiting the LP relaxation symmetry groups of various formulations.

In Section 5.2, some open problems resulting from the research in this dissertation are presented. In Section 5.3, several conjectures are made based on the Chapter 4 computational research results. In Section 5.4, several follow-up computational projects are proposed.

1.3 Organization of Dissertation

This dissertation is organized into five chapters and two appendices. Chapter 2 provides a review of recent publications that have directly lead to the study of the

research problem discussed in this dissertation. Chapter 3 discusses theoretical work derived from the literature review. Chapter 4 discusses the four computational research objectives. Chapter 5 provides a list of conjectures and research directions for future efforts. Appendix A contains part of the computer code that was used to achieve the results in Chapters 2 and 3. Finally, Appendix B contains the computer code that was used to achieve the computational research results in Chapter 4.

II. Literature Review

2.1 Chapter Introduction

This chapter first discusses various methods that have been developed to generate OAs using ILPs. It then presents background material on permutation groups. This is followed by an explanation of the relationship between permutation groups as symmetry groups of OA-generating ILPs and isomorphisms of OAs. This leads to a discussion of how to find large subgroups of the symmetry groups of ILPs more efficiently. The chapter concludes with a summary of the solver developed by Margot [22] that takes advantage of symmetry groups of ILPs to more rapidly enumerate solutions.

2.2 ILPs That Generate OAs

One efficient and effective approach to generating OAs is to formulate an ILP whose feasible set is the set of all OA(N, k, s, t). If the ILP has no feasible solution, then an OA(N, k, s, t) does not exist.

Rosenberg [29] proved the following Lemma.

Lemma 1. *Define a sequence* $\{a_c\}$ *recursively by*

$$a_0 = \lambda, \quad a_c = \lambda - \sum_{e=0}^{c-1} a_e \binom{k-t}{c-e} (s-1)^{c-e} \quad \text{for } c \geq 1. \tag{2.1}$$

Let $\mathbf{z}$, $\mathbf{x}$ *and* $\mathbf{y} \in \{1, \cdots, s\}^k$ *be row vectors with* $0 \leq d(\mathbf{z}, \mathbf{x}) \leq t$ *where* $d(\mathbf{z}, \mathbf{x})$ *is the number of non-zero entries in* $\mathbf{z} - \mathbf{x}$. *Also, let* $I_{\mathbf{x}} = \{i \in \{1, \cdots, k\} : x_i \neq z_i\}$ *and* $J_{\mathbf{x}} = \{\mathbf{y} \in \{1, \cdots, s\}^k : y_i = x_i \ \ \forall i \in I_{\mathbf{x}}\}$. *Then*

$$N_{\mathbf{x}} = a_{t-d(\mathbf{z},\mathbf{x})} + (-1)^{t-d(\mathbf{z},\mathbf{x})+1} \sum_{\substack{\mathbf{y} \in J_{\mathbf{x}} \\ d(\mathbf{z},\mathbf{y}) > t}} \binom{d(\mathbf{z},\mathbf{y}) - d(\mathbf{z},\mathbf{x}) - 1}{t - d(\mathbf{z},\mathbf{x})} N_{\mathbf{y}},$$
$$N_{\mathbf{y}} \geq 0, \text{ for } \mathbf{y} \text{ such that } d(\mathbf{z},\mathbf{y}) > t, \tag{2.2}$$

where $N_{\mathbf{x}}$, $N_{\mathbf{y}}$ are the number of times factor level combinations $\mathbf{x}$, $\mathbf{y}$ *appear in a conjectured* $OA(\lambda s^t, k, s, t)$.

If we are given the values of $N_{\mathbf{x}}$ for every $\mathbf{x} \in \{1, 2, ..., s\}^k$ where $d(\mathbf{x}, \mathbf{z}) > t$, then equation (2.2) has at most one integer solution with $N_{\mathbf{x}} \geq 0$ for the remaining $N_{\mathbf{x}}$'s. The system of linear equations in equation (2.2) can be viewed as the constraints of an ILP with equality constraints, where the objective function $\mathbf{c}^T\mathbf{x}$ is chosen to be zero. All feasible solutions of this ILP are the indicator vectors of all OA(N, k, s, t).

In a separate effort, Appa *et al.* [3] examined two different ways to use a combination of constraint and integer programming to find $p \times p$ Orthogonal Latin Squares (OLS) for $3 \leq p \leq 12$ and three $p \times p$ Mutually Orthogonal Latin Squares (MOLS). In terms of the notation used here, k $p \times p$ MOLS correspond to an OA$(p^2, k + 2, p, 2)$. Appa *et al.* [2] successfully used an ILP formulation to generate $p \times p$ OLS's for $4 \leq p \leq 5$ and $7 \leq p \leq 12$. Additionally, they proved that 6×6 OLS do not exist.

Without the knowledge of Lemma 1 in Rosenberg [29], Bulutoglu and Margot [7] used an ILP formulation to enumerate OAs. They defined $\mathbf{D}_{s^k}$ to be the full factorial design where each run is treated as a base-s integer and placed in lexicographical order. They then defined the *indicator vector* $\mathbf{x}$ where every x_i is the number of times the ith run of $\mathbf{D}_{s^k}$ appears in a particular k-factor, s-level fractional factorial design $\mathbf{D}$. Next, they defined a set T of $s^t \times s^k$ matrices obtained by taking the Kronecker products of t $\mathbf{I}_s$'s and $(k - t)\mathbf{1}_s^T$'s in every possible order, where $\mathbf{I}_s$ represents the $s \times s$ identity matrix and $\mathbf{1}_s$ is an $s \times 1$ vector of 1's. As such, $|T| = \binom{k}{t}$ and every matrix $\mathbf{A}_j \in T$ has the form $\mathbf{A}_j = \mathbf{M}_{1j} \otimes \mathbf{M}_{2j} \otimes ... \otimes \mathbf{M}_{kj}$ where $1 \leq j \leq |T|$ and $\mathbf{M}_{ij} \in \{\mathbf{1}_s^T, \mathbf{I}_s\}$. Finally, they observed that for the set $L_j = \{i_{1j}, i_{2j}, ..., i_{tj} \mid \mathbf{M}_{ij} = \mathbf{I}_s\}$, $\mathbf{A}_j\mathbf{x} = \lambda\mathbf{1}_{s^t}$ if and only if when $\mathbf{D}$ is projected onto the factors indexed by L_j, all s^t factor level combinations appear exactly λ times. This means that a solution to

$$\mathbf{A}_j\mathbf{x} = \lambda\mathbf{1}_{s^t} \text{ for } \mathbf{x} \geq 0 \text{ for } j = 1, 2, ..., \binom{k}{t}$$

is integral if and only if OA(N, k, s, t) exists and that each integer solution identifies an OA(N, k, s, t) [7].

Let $r_{\max}$ be the maximum number of times a factor level combination can appear in an OA(N, k, s, t) then Lemma 5 in Bulutoglu and Margot [7] implies $r_{\max} \leq p_{\max}$, where

$$\begin{aligned} & p_{\max} = \max \quad Y_0 \\ \text{s.t.} \quad & \sum_{i=h}^{k} \binom{i}{k} Y_{k-i} = \lambda s^{t-h} \binom{k}{h} \quad \text{for} \quad 0 \leq h \leq t, \\ & Y_i \in \mathbb{Z}_{\geq 0} \quad \text{for} \quad 0 \leq i \leq k. \end{aligned} \tag{2.3}$$

The computer code that they used to find $p_{\max}$ was also used in this dissertation.

Letting $\mathbf{1}_{\binom{k}{t}s^t}$ be the $\binom{k}{t}s^t \times 1$ vector of 1's and $\mathbf{A}(k, s, t)$ be the $\binom{k}{t}s^t \times s^k$ matrix

$$\mathbf{A}(k, s, t) = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \\ \mathbf{A}_{\binom{k}{t}} \end{bmatrix},$$

then Theorem 2 in Bulutoglu and Margot [7] implies that the set of all solutions to the ILP

$$\begin{aligned} \min \quad & \mathbf{1}_{s^k}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}(k, s, t)\mathbf{x} = \lambda \mathbf{1}_{\binom{k}{t}s^t} \\ & \mathbf{x} \in \{0, 1, 2, ..., p_{\max}\}^{s^k} \end{aligned} \tag{2.4}$$

is precisely the set of indicator vectors for all OA(N, k, s, t). In ILP (2.4) the objective function $\mathbf{1}_{s^k}^T \mathbf{x}$ was introduced only for convenience and is equal to N for every feasible solution.

Bulutoglu and Ryan [8] provided a more efficient version of this ILP that has only $\sum_{q=0}^{t}(s-1)^q\binom{k}{q}$ equality constraints with no redundant constraints:

$$\begin{aligned}
\min \quad & 0 \\
\text{s.t.} \quad & \sum_{\substack{(i_1,\ldots,i_k)\in\{0,\ldots,s-1\}^k \\ (i_{j_1},\ldots,i_{j_q})\in\{0,\ldots,s-2\}^q}} x_{\left[\sum_{j=1}^{k} i_j s^{k-j}+1\right]} = \frac{N}{s^q} \quad \begin{array}{l}\text{for } q=0,\ldots,t \text{ and} \\ \text{each subset of } q \\ \text{indices } \left\{i_{j_1}, i_{j_2}, \ldots, i_{j_q}\right\},\end{array} \\
& x_1 \geq 1, \\
& \mathbf{x} \in \{0,1,2,\ldots,p_{\max}\}^{s^k}.
\end{aligned} \tag{2.5}$$

The constraint $x_1 \geq 1$ in ILP (2.5) ensures that $(0, 0, ..., 0)$ appears at least once and helps the solver find feasible solutions faster [8]. Using 0 as the objective function also speeds up the enumeration process. The algorithm in Figure 2.1 summarizes how to utilize ILP (2.5) to enumerate all OA(N, k, s, t).

1. Input N, k, s and t. Let $\mathbf{T} = \emptyset$.
2. Determine $p_{\max}$ using ILP (2.3).
3. Find the set of all feasible solutions $\mathbf{X}$ to ILP (2.5).
4. For each solution $\mathbf{x}_q \in \mathbf{X}$,
   (a) Let the factor level combination $(i_1, ..., i_k)$ appear in $\mathbf{T}_q$ $x_{\left[\sum_{j=1}^{k} i_j s^{k-j}+1\right],q}$ many times.
   (b) Set $\mathbf{T} = \mathbf{T} \cup \mathbf{T}_q$.
5. Return $\mathbf{T}$.

Figure 2.1: Improved Bulutoglu and Margot ILP [8] Based Algorithm

An additional formulation for strength-2 OAs arises from Hadamard matrices. A *Hadamard matrix* is an orthogonal $N \times N$ matrix with entries that are either -1 or $+1$.

For any $n \in \mathbb{Z}_{\geq 0}$, OA$(4n, 4n-1, 2, 2)$ exist if and only if there exists a Hadamard matrix of order $4n$ [13]. This is true because deleting the column of all $+1$'s from a $N \times N$ Hadamard matrix, results in an OA$(N, N-1, 2, 2)$ where the levels are coded as -1 and $+1$ (Hadamard matrices exist only if N is divisible by 4).

Williamson [34] defined *Williamson matrices* to be any four $n \times n$ circulant matrices $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$ and $\mathbf{W}_4$ with entries equal to -1 or $+1$ such that

$$\mathbf{W}_1^2 + \mathbf{W}_2^2 + \mathbf{W}_3^2 + \mathbf{W}_4^2 = 4n\mathbf{I}_n.$$

Using the Williamson matrices, a $4n \times 4n$ Hadamard matrix $\mathbf{H}_{4n}$ is constructed as

$$\mathbf{H}_{4n} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ -\mathbf{W}_2 & \mathbf{W}_1 & \mathbf{W}_4 & -\mathbf{W}_3 \\ -\mathbf{W}_3 & -\mathbf{W}_4 & \mathbf{W}_1 & \mathbf{W}_2 \\ -\mathbf{W}_4 & \mathbf{W}_3 & -\mathbf{W}_2 & \mathbf{W}_1 \end{bmatrix}. \tag{2.6}$$

Let ω^j be the primitive jth root of unity and $\omega_j = \omega^j + \omega^{n-j}$. Williamson [34] showed that Williamson matrices exist if and only if

$$T_1^2 + T_2^2 + T_3^2 + T_4^2 = 4n$$

where each T_i has the form $1 \pm 2\omega_{j_1} \pm 2\omega_{j_2} \pm ...$ and each ω_j appears only once in only one member of the set $\{T_1, T_2, T_3, T_4\}$. Noting that $\omega^j = \cos\left(\frac{2j\pi}{n}\right) + \sqrt{-1}\sin\left(\frac{2j\pi}{n}\right)$ leads to the following theorem:

Theorem 1. *Let $\sum_{k=1}^{4} \mu_{k,l}^2 = 4n$ where n is an odd integer, $l \mid n$, $\mu_{k,l} \neq 0$, and $\mu_{k,l} = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 4\cos\left(\frac{2\pi il}{n}\right)(t_{i2k} - t_{i1k})$. Williamson matrices of order n exist if and only if there is at least one feasible solution to the following system of equations in t_{ijk}*

$$\begin{aligned} &\sum_{k=1}^{4} \mu_{k,l}^2 = \sum_{k=1}^{4} \left(\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 4\cos\left(\frac{2\pi il}{n}\right)(t_{i2k} - t_{i1k}) \right)^2 = 4n \text{ for } l \mid n, \\ &\sum_{k=1}^{4} \sum_{j=1}^{2} t_{ijk} = 1 \quad \text{for } i = 1, 2, ..., \left\lfloor \frac{n}{2} \right\rfloor, \\ &t_{ijk} \in \{0, 1\} \quad \text{for } i = 1, 2, ..., \left\lfloor \frac{n}{2} \right\rfloor,\ j = 1, 2 \text{ and } k = 1, 2, 3, 4. \end{aligned} \tag{2.7}$$

The nonlinear constraints in constraints (2.7) can be linearized as follows. First, using $t_{ijk}^2 = t_{ijk}$, expand the constraints:

$$\begin{aligned} 4n &= \sum_{k=1}^{4} \mu_{k,l}^2 = \sum_{k=1}^{4} \left(\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 4\cos\left(\frac{2\pi i l}{n}\right)(t_{i2k} - t_{i1k}) \right)^2 \\ &= 16 \sum_{k=1}^{4} \left(\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \cos^2\left(\frac{2\pi i l}{n}\right)(t_{i2k} + t_{i1k} - 2t_{i2k}t_{i1k}) \right) \\ &+ 32 \sum_{k=1}^{4} \left(\sum_{\substack{i'<i'' \\ i'' \le \lfloor \frac{n}{2} \rfloor}} \cos\left(\frac{2\pi i' l}{n}\right)\cos\left(\frac{2\pi i'' l}{n}\right)(t_{i'2k}t_{i''2k} - t_{i'2k}t_{i''1k} - t_{i'1k}t_{i''2k} + t_{i'1k}t_{i''1k}) \right) \end{aligned} \tag{2.8}$$

Next, let $y_{(i_1 j_1 k_1)(i_2 j_2 k_2)} = t_{i_1 j_1 k_1} t_{i_2 j_2 k_2}$, add constraints $0 \le t_{i_1 j_1 k_1} + t_{i_2 j_2 k_2} - 2y_{(i_1 j_1 k_1)(i_2 j_2 k_2)} \le 1$ for each $y_{(i_1 j_1 k_1)(i_2 j_2 k_2)}$ and substitute the $y_{(i_1 j_1 k_1)(i_2 j_2 k_2)}$'s into equalities (2.8). The nonlinear system of equations (2.7) is now written as the linear system of constraints (2.9):

$$\begin{aligned} 4n &= 16 \sum_{k=1}^{4} \left(\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \cos^2\left(\frac{2\pi i l}{n}\right)(t_{i2k} + t_{i1k} - 2y_{(i2k)(i1k)}) \right) \\ &+ 32 \sum_{k=1}^{4} \left(\sum_{\substack{i'<i'' \\ i'' \le \lfloor \frac{n}{2} \rfloor}} \cos\left(\frac{2\pi i' l}{n}\right)\cos\left(\frac{2\pi i'' l}{n}\right)(y_{(i'2k)(i''2k)} - y_{(i'2k)(i''1k)} - y_{(i'1k)(i''2k)} + y_{(i'1k)(i''1k)}) \right) \end{aligned}$$

for $l \mid n$,

$$\begin{aligned} &2y_{(i_1 j_1 k_1)(i_2 j_2 k_2)} - t_{i_1 j_1 k_1} - t_{i_2 j_2 k_2} \le 0 \\ &t_{i_1 j_1 k_1} + t_{i_2 j_2 k_2} - 2y_{(i_1 j_1 k_1)(i_2 j_2 k_2)} \le 1 \end{aligned} \quad \text{for each } y_{(i_1 j_1 k_1)(i_2 j_2 k_2)},$$

$$\sum_{k=1}^{4} \sum_{j=1}^{2} t_{ijk} = 1 \quad \text{for } i = 1, 2, ..., \left\lfloor \frac{n}{2} \right\rfloor,$$

$$y_{(i_1 j_1 k_1)(i_2 j_2 k_2)} \in \{0, 1\},\ t_{ijk} \in \{0, 1\} \quad \text{for } i = 1, 2, ..., \left\lfloor \frac{n}{2} \right\rfloor,\ j = 1, 2 \text{ and } k = 1, 2, 3, 4. \tag{2.9}$$

Hence, the system of quadratic constraints (2.8) with $4n - 4$ binary variables is turned into a linear system of constraints with $4n - 4 + 4\left\lfloor \frac{n}{2} \right\rfloor + 4\binom{\lfloor \frac{n}{2} \rfloor}{2} = (n^2 - 1)/2 + 4n - 4$ binary variables. Combining Theorem 1 and Example 9.1 in Seberry and Yamada [31] leads to the algorithm in Figure 2.2.

1. Let $n \in \mathbb{Z}_{\geq 0}$, $N = 4n$, and $\frac{n}{2} \notin \mathbb{Z}_{\geq 0}$. Let $\mathbf{U}$ be an empty set of OA$(N, N-1, 2, 2)$.

2. Find all solutions to the Binary Integer Linear Program (BILP) (using constraints (2.9)):

$$
\begin{aligned}
&\min 0 \\
&\text{s.t. } 4n = 16 \sum_{k=1}^{4} \left(\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \cos^2\left(\frac{2\pi i l}{n}\right) \left(t_{i2k} + t_{i1k} - 2y_{(i2k)(i1k)}\right) \right) \\
&\quad + 32 \sum_{k=1}^{4} \left(\sum_{\substack{i'<i'' \\ i'' \leq \lfloor \frac{n}{2} \rfloor}} \cos\left(\frac{2\pi i' l}{n}\right) \cos\left(\frac{2\pi i'' l}{n}\right) \left(y_{(i'2k)(i''2k)} - y_{(i'2k)(i''1k)} - y_{(i'1k)(i''2k)} + y_{(i'1k)(i''1k)}\right) \right) \quad \text{for } l \mid n, \\
&\quad 2y_{(i_1 j_1 k_1)(i_2 j_2 k_2)} - t_{i_1 j_1 k_1} - t_{i_2 j_2 k_2} \leq 0 \\
&\quad t_{i_1 j_1 k_1} + t_{i_2 j_2 k_2} - 2y_{(i_1 j_1 k_1)(i_2 j_2 k_2)} \leq 1 \quad \text{for each } y_{(i_1 j_1 k_1)(i_2 j_2 k_2)}, \\
&\quad \sum_{k=1}^{4} \sum_{j=1}^{2} t_{ijk} = 1 \quad \text{for } i = 1, 2, ..., \left\lfloor \frac{n}{2} \right\rfloor, \\
&\quad y_{(i_1 j_1 k_1)(i_2 j_2 k_2)} \in \{0, 1\},\ t_{ijk} \in \{0, 1\} \quad \text{for } i = 1, 2, ..., \left\lfloor \frac{n}{2} \right\rfloor,\ j = 1, 2 \text{ and } k = 1, 2, 3, 4.
\end{aligned}
\tag{2.10}
$$

3. Let $\mathbf{Y}$ be the set of all solutions to ILP (2.10) and $\mathbf{t}_q = \left(t_{ijkq}\right) \in \mathbf{Y}$ be the qth solution to ILP (2.10). Let $\mathbf{W}_{kq}$ for $k = 1, 2, 3, 4$ be the set of Williamson matrices calculated from the solution $\mathbf{Y}_q$. Also, let $w_{k,j,q}$ denote the j^{th} element of the first row of $\mathbf{W}_{kq}$. Finally, let $\mathbf{H}_q$ be the $4n \times 4n$ Hadamard matrix calculated from the $\mathbf{W}_{kq}$'s.

4. For each $\mathbf{t}_q \in \mathbf{Y}$

   (a) Set $w_{k,1,q} = 1$, $w_{k,j,q} = t_{(j-1),2,k,q} - t_{(j-1),1,k,q}$ and $w_{k,n-j+2,q} = w_{k,j,q}$ for $j = 2, 3, ... \left\lfloor \frac{n}{2} \right\rfloor + 1$ and $k = 1, 2, 3, 4$.

   (b) For each $k = 1, 2, 3, 4$, $\mathbf{W}_{kq} = \begin{bmatrix} w_{k,1,q} & w_{k,2,q} & \cdots & w_{k,n,q} \\ w_{k,n,q} & w_{k,1,q} & \cdots & w_{k,n-1,q} \\ w_{k,n-1,q} & w_{k,n,q} & \cdots & w_{k,n-2,q} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,2,q} & w_{k,3,q} & \cdots & w_{k,1,q} \end{bmatrix}$.

   (c) Use equation (2.6) to calculate $\mathbf{H}_q$ using the $\mathbf{W}_{kq}$'s found in previous step. Update $\mathbf{H}_q$ by removing its first column and set $\mathbf{U} = \mathbf{U} \cup \left\{\mathbf{H}_q\right\}$.

5. Return $\mathbf{U}$.

Figure 2.2: Algorithm to Generate OA$(N, N-1, 2, 2)$ from Williamson Matrices

The most efficient ILP formulations for enumerating all OA(N, k, s, t) extend all OA(N, i, s, t) to all OA$(N, i+1, s, t)$ for $t \leq i \leq k-1$. Bulutoglu and Ryan [8] demonstrated a BILP formulation that extends a given OA$(N, k-1, s, t)$ to all possible OA(N, k, s, t). Their BILP formulation is based on the following lemma in [8]

Lemma 2. *Let* $\mathbf{D}$ *be a* N*-run,* k*-factor,* s*-level factorial design with columns* $\{\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_k\}$. *Let* $\mathbf{D}'$ *be the* $N \times (s-1)\,k$ *matrix with columns* $\{\boldsymbol{d}'_1, \boldsymbol{d}'_2, ..., \boldsymbol{d}'_{(s-1)k}\}$ *where for* $j = 1, 2, ..., k$ *and* $r = 1, 2, ..., s-1$ *the* i*th entry of* $\boldsymbol{d}'_{(s-1)(j-1)+r}$ *is* 1 *if the* i*th entry of* $\boldsymbol{d}_j$ *is* $r-1$ *and is* 0 *otherwise. Then* $\mathbf{D}$ *is an OA*(N, k, s, t) *if and only if*

$$\sum_{i=1}^{N} d'_{ih_1} d'_{ih_2} \cdots d'_{ih_q} = \frac{N}{s^q} \quad \text{for } q = 1, 2, \cdots, t, \tag{2.11}$$

and for all subset of size q *columns* $\{\boldsymbol{d}'_{h_1}, \boldsymbol{d}'_{h_2}, ..., \boldsymbol{d}'_{h_q}\}$ *of* $\mathbf{D}'$ *such that* $\lceil h_{i'}/(s-1) \rceil \neq \left\lceil h_{j'}/(s-1) \right\rceil \ \forall\ 1 \leq i' < j' \leq (s-1)\,k$.

Let $\mathbf{D}'$ be a solution to the system of equations (2.11). Let the jth column of $\mathbf{D}$, $\mathbf{D}_j$ be

$$\mathbf{D}_j = \sum_{r=1}^{s-1} (r-1)\mathbf{d}'_{(s-1)(j-1)+r} + (s-1)\left(\mathbf{1}_N - \sum_{r=1}^{s-1} \mathbf{d}'_{(s-1)(j-1)+r}\right)$$

for $j = 1, 2, ..., k$ [8]. Then, $\mathbf{D}$ is the corresponding OA(N, k, s, t) obtained from $\mathbf{D}'$. For a known N-run, $(k-1)$-factor, s-level OA$(N, k-1, s, t)$ $\mathbf{D}$, Bulutoglu and Ryan [8] showed that all OA(N, k, s, t) extensions of $\mathbf{D}$ can be constructed by utilizing a BILP via the algorithm described in Figure 2.3. All possible extensions of all OA$(N, k-1, s, t)$ to all OA(N, k, s, t) are obtained by implementing the algorithm in Figure 2.3 with each OA$(N, k-1, s, t)$ $\mathbf{D}$ as input.

1. Follow Lemma 2 to construct $\mathbf{D}'$ from input $\mathbf{D}$.

2. Append to $\mathbf{D}'$ $s-1$ columns of binary variables $\mathbf{x}_r = (x_{1r}, x_{2r}, ..., x_{Nr})^T$ for each $r = 1, 2, ..., s-1$. (The system of equations (2.11) is linear with binary variables.)

3. For each pair of replicated run indices $1 \le i'' < j'' \le N$ in the input design and for any $q-1$ columns $\mathbf{d}'_{h_1}, \mathbf{d}'_{h_2}, ..., \mathbf{d}'_{h_{q-1}}$ as in Lemma 2 with the last $(s-1)$ columns of $\mathbf{D}$ deleted, find all feasible solutions to the BILP

$$
\begin{aligned}
\min \quad & 0 \\
\text{s.t.} \quad & \sum_{i=1}^{N} x_{ir} = \frac{N}{s} && \text{for } r = 1, 2, ..., s-1, \\
& \sum_{i=1}^{N} d'_{ih_1} d'_{ih_2} \cdots d'_{ih_{q-1}} x_{ir} = \frac{N}{s^q} && \text{for } q = 2, ..., t \text{ and } r = 1, 2, ..., s-1, \\
& \sum_{r=1}^{s-1} x_{ir} \le 1 && \text{for } i = 1, 2, ..., N, \\
& \sum_{m=1}^{r} \left(x_{i''m} - x_{j''m} \right) \ge 0 && \text{for } r = 1, 2, ..., s-1, \\
& x_{1,1} = 1, \\
& \mathbf{x}_r \in \{0, 1\}^N && \text{for } r = 1, 2, ..., s-1.
\end{aligned}
\tag{2.12}
$$

4. Each solution matrix $[\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{s-1}]$ to BILP (2.12) corresponds to an OA(N, k, s, t) extension of the input OA$(N, k-1, s, t)$ and every extension corresponds to a solution matrix.

Figure 2.3: Identity Group Algorithm from Bulutoglu and Ryan [8]

Bulutoglu and Ryan [8] combined the ILP formulations in Figures 2.1 and 2.3 to create another formulation that extends all OA$(N, k-1, s, t)$ to all OA(N, k, s, t) as in the above algorithm. This new formulation reduces the number of variables from s^k in ILP (2.5) to hs, where h is the number of distinct runs in the extended OA$(N, k-1, s, t)$. This additional extension formulation is described in Figure 2.4.

1. Follow Lemma 2 to construct $\mathbf{D}'$ from input $\mathbf{D}$. Let r_i for $i = 1, 2, \cdots, s^{k-1}$ be the number of times the ith row of $\mathbf{D}_s^{k-1}$ appears in the input design, where $\mathbf{D}_s^{k-1}$ is all the s^{k-1} level combinations with $k-1$ factors ordered lexicographically. Set h equal to the number of distinct runs in $\mathbf{D}$. Let $1 \le i_1 \le i_2 \le \cdots \le i_h \le s^{k-1}$ be such that $r_{i_l} > 0$ for $l = 1, 2, \cdots, h$.

2. Define $1 = p_1 \le p_2 \le ... \le p_h \le N$ such that the p_{l+1}th row is the first row that differs from the p_lth row. Also, let $\mathbf{x}$ be the indicator vector for the desired OA(N, k, s, t) formatted as a solution to ILP (2.5) and $\mathbf{x}^{h(s-1)}$ be $\mathbf{x}$ with all of the entries that do not appear in the constraint matrix of ILP (2.13) deleted. This leaves the vector $\mathbf{x}^{h(s-1)}$ with $h(s-1)$ entries.

3. For any $q-1$ columns $\mathbf{d}'_{\alpha_1}, \mathbf{d}'_{\alpha_2}, ..., \mathbf{d}'_{\alpha_{q-1}}$ as in Lemma 2 with the last $(s-1)$ columns of $\mathbf{D}'$ deleted, find all solutions to ILP (2.13).

$$
\begin{aligned}
\min \quad & 0 \\
\text{s.t.} \quad & \sum_{l=1}^{h} x_{(i_l-1)s+j} = \lambda s^{t-1}, \\
& \sum_{l=1}^{h} d'_{p_l\alpha_1} d'_{p_l\alpha_2} \cdots d'_{p_l\alpha_{q-1}} x_{(i_l-1)s+j} = \lambda s^{t-q}, \\
& \sum_{j=1}^{s-1} x_{(i_l-1)s+j} \le r_{i_l}, \\
& x_1 \ge 1, \\
& \mathbf{x}_{(i_l-1)s+j} \in \{0, 1, ..., \min(r_{i_l}, p_{\max})\}, \\
& \text{for } q = 2, ..., t,\ j = 1, 2, ..., s-1,\ l = 1, 2, ..., h \text{ and} \quad \lceil \alpha_{i'}/(s-1) \rceil \ne \left\lceil \alpha_{j'}/(s-1) \right\rceil \\
& \forall \quad 1 \le i' < j' \le (s-1)(k-1).
\end{aligned}
\tag{2.13}
$$

4. Each solution $\mathbf{x}^{h(s-1)}$ corresponds to an OA(N, k, s, t) extension of the input OA$(N, k-1, s, t)$ and every such extension corresponds to a solution.

Figure 2.4: Extension to Improved Bulutoglu and Margot ILP [8]

Permuting the levels (symbols) of a factor in a factorial design is called a *level permutation*. Two k-factor factorial designs are called *isomorphic* if one can be obtained from the other by permuting its columns, rows, and applying level permutations in its

factors. For a given s-level, k-factor, N-run factorial design $\mathbf{D}$, Bulutoglu and Ryan [8] defined a graph $G(\mathbf{D})$. They showed that $\mathbf{D}_1$ and $\mathbf{D}_2$ are isomorphic designs if and only if $G(\mathbf{D}_1)$ and $G(\mathbf{D}_2)$ are isomorphic graphs. The following steps extend the algorithm in Figure 2.4 to enumerate all non-isomorphic OA(N, k, s, t) for $t+1 \leq k \leq k_{\max}(N, s, t)$, where $k_{\max}(N, s, t)$ is the maximimum number of factors such that an OA$(N, k_{\max}(N, s, t), s, t)$ exists.

5. If there are no solutions, STOP.
6. Convert each OA(N, k, s, t) from Step 4 to the graph defined by Bulutoglu and Ryan [8]. Let Ω be the set of resulting graphs.
7. Reduce Ω to a set of nonisomorphic graphs by using Nauty [25].
8. Retain only the OA(N, k, s, t)'s that correspond to the retained graphs.
9. Set $k = k + 1$ and GO TO Step 1.

Figure 2.5: Additional Steps to Figure 2.4 to Enumerate Non-Isomorphic OA(N, k, s, t) [8].

For $s = 2$, the following definitions and theorem were used by Bulutoglu [6] to extend the algorithm in Figure 2.4 to find the maximum number of factors, $k_{\max}$ such that an OA$(N, k_{\max}, 2, t)$ exists.

Definition 1. *Let* $\mathbf{Y}_1$ *and* $\mathbf{Y}_2$ *be* 2*-level, k-factor factorial designs whose levels are* ± 1. *Then,* $\mathbf{Y}_1$ *and* $\mathbf{Y}_2$ *are called* Hadamard equivalent *if* $\mathbf{Y}_2$ *can be obtained from* $\mathbf{Y}_1$ *by applying signed permutations to the columns and/or rows of* $\mathbf{Y}_1$ *[24].*

Definition 2. *Let* $\mathbf{X}_1$ *and* $\mathbf{X}_2$ *be* 2*-level, k-factor factorial designs whose levels are* ± 1. *Then,* $\mathbf{X}_1$ *and* $\mathbf{X}_2$ *are called* Orthogonal Design (OD) equivalent *if* $[\mathbf{1}\ \mathbf{X}_2]$ *is Hadamard equivalent to* $[\mathbf{1}\ \mathbf{X}_1]$ *[6].*

Bulutoglu [6] showed the following result:

Theorem 2. *Let $k_1 < k_2$ and $\mathbf{X}_1$ be an OA$(N, k_1, 2, t)$ that extends to an OA$(N, k_2, 2, t)$. If $\mathbf{X}_2$ is an OA$(N, k_1, 2, t)$ that is OD equivalent to $\mathbf{X}_1$, then $\mathbf{X}_2$ also extends to an OA$(N, k_2, 2, t)$.*

For a 2-level, k-factor, N-run design $\mathbf{Y}_1$, McKay [24] defined a graph $X(\mathbf{Y}_1)$ with $2(k + N)$ vertices and showed that $\mathbf{Y}_1$ is Hadamard equivalent to $\mathbf{Y}_2$ if and only if $X(\mathbf{Y}_1)$ and $X(\mathbf{Y}_2)$ are isomorphic graphs.

For $s = 2$, the algorithm in Figure 2.5 is modified to find a set of all non-OD equivalent OA$(N, k, 2, t)$ for $k \leq k_{\max}(N, 2, t)$ by replacing Step 6 with the following step:

> 6. Add a column of 1's to each OA$(N, k, 2, t)$ from Step 4. Convert each resulting design to the graph defined by McKay [24]. Let Ω be the set of resulting graphs.

Figure 2.6: New Step 6 in Figure 2.5 Algorithm to Find $k_{\max}(N, 2, t)$ [6].

The modified Figure 2.5 algorithm is faster as it requires solving much fewer ILPs at each extension step. Also, a set of all non-isomorphic OA$(N, k, 2, t)$ can be extracted from a set of non-OD equivalent OA$(N, k, 2, t)$. Hence, the modified Figure 2.5 algorithm can be used to enumerate all non-isomorphic OA$(N, k, 2, t)$ for $k \leq k_{\max}(N, 2, t)$.

These are just a few different ways that ILPs are used to enumerate OAs. The next section defines the symmetry group of an ILP.

2.3 Symmetry Groups in ILPs

In order to study the symmetries of an ILP, some definitions are introduced. $I^n = \{i \in \mathbb{Z}_{>0} \mid i \leq n\}$ is the *ground set* of size n. Let S_n be the set of all permutations of the elements in I^n. Let π_i be the image of $i \in I^n$ under the permutation π. The *identity permutation* I is such that $\pi_i = i \quad \forall\ i \in I^n$. For any $\mathbf{v} \in \mathbb{R}^n$, let $\mathbf{w} = \pi(\mathbf{v})$ indicate the vector $\mathbf{w} \in \mathbb{R}^n$ that results from applying the permutation π to $\mathbf{v}$. For any two permutations $\pi^1, \pi^2 \in S_n$, the *composition* of these two permutations is $\pi^1(\pi^2)$. The common notation

for this is $\pi^1 \cdot \pi^2$. The identity for the composition of permutations is the aforementioned identity permutation I, where $\pi \cdot I = I \cdot \pi = \pi \quad \forall\ \pi \in S_n$. For any $\pi^1, \pi^2, \pi^3 \in S_n$, $(\pi^1 \cdot \pi^2) \cdot \pi^3 = \pi^1 \cdot (\pi^2 \cdot \pi^3)$ *i.e.* the composition of permutations is associative [23].

The subset $G \subseteq S_n$ is called a *group* if it has the following properties:

1. $I \in G$.

2. $\pi^1 \cdot \pi^2 \in G \quad \forall\ \pi^1, \pi^2 \in G$.

3. For all $\pi \in G$, there exists a unique inverse $\pi^{-1} \in G$ such that $\pi \cdot \pi^{-1} = \pi^{-1} \cdot \pi = I$.

The number of permutations in G is called the *order* of the group, denoted by $|G|$. For example, $|S_n| = n!$. A *subgroup* is a subset of G that is also itself a group. All groups in this dissertation have finite order, as their elements are permutations of finite sets [23].

Margot [23] defined the *symmetry group* G of an ILP as

$$G = \{\pi \in S_n \mid \mathbf{c}^T\mathbf{x} = \mathbf{c}^T\pi(\mathbf{x}) \text{ and } \pi(\mathbf{x}) \in \mathcal{F} \quad \forall\ \mathbf{x} \in \mathcal{F}\},$$

where $\mathcal{F}$ is the set of all feasible solutions to the ILP. Elements of a set $\Omega \subseteq S_n$ which can be composed in some way some number of times to obtain each permutation in the group $G \subseteq S_n$, are called *generators* of the group G, and G is said to be *generated* by Ω. A set of $O(n^2)$ generators is all that is needed to generate any subgroup of S_n [23]. Additionally, the *stabilizer* of a vector $\mathbf{v} \in \mathbb{R}^n$ under G is

$$\text{stab}(\mathbf{v}, G) = \{\pi \in G \mid \mathbf{v} = \pi(\mathbf{v})\}$$

and the *orbit* of a vector $\mathbf{v} \in \mathbb{R}^n$ under G is

$$\text{orb}(\mathbf{v}, G) = \{\mathbf{w} \in \mathbb{R}^n \mid \mathbf{w} = \pi(\mathbf{v}) \text{ for at least one } \pi \in G \subseteq S_n\}.$$

The *restriction* of G to J is the set of all permutations of just the elements of a nonempty set $J \subseteq \{1, 2, ..., n\}$ that are induced by the permutations in $\text{stab}(J, G)$. The restriction is itself a group. Finally, $\mathbf{v} \in \mathbb{R}^n$ is *lexicographically smaller* (*larger*) than $\mathbf{w} \in \mathbb{R}^n$ if $v_j = w_j$ and $v_k < w_k$ ($v_k > w_k$) for some k with $1 \leq j < k$ and $1 \leq k \leq n$. The notation for this is $\mathbf{v} <_L \mathbf{w}$ ($\mathbf{v} >_L \mathbf{w}$) [23].

The definitions provided in this section provide a framework for the discussions in the next section. Next section discusses how symmetry groups in OA-generating ILPs relate to isomorphism of OAs.

2.4 Symmetry in ILPs and Isomorphism of OAs

As previously discussed, OAs are known to be universally optimal for estimating certain statistical models [35]. It is well known that isomorphic designs have the same GWP. Thus, the GMA property is invariant between isomorphic designs. It is often quite difficult to determine whether an OA(N, k, s, t) exists for a given N, k, s, t combination. However, if one OA(N, k, s, t) is found, it is just a matter of permuting symbols, rows and/or columns to find other OAs that are isomorphic to this OA.

Additionally, since the ILPs of interest are constructed to generate OAs that are isomorphic to many other OAs, these ILPs all contain variables that can be permuted without changing the problem. Any ILP that contains at least some variables that can be permuted without changing the feasibility and optimality of its solutions is said to be *symmetric*. Many classical problems in operations research, combinatorics and statistical experimental design are formulated as symmetric ILPs. In addition to generating orthogonal arrays, symmetric ILPs are frequently encountered in job scheduling, code construction, covering design construction and graph coloring problems [23].

There are a number of commercial and academic solvers available that are designed to find all optimal solutions to ILPs. The Branch-and-Bound and Branch-and-Cut algorithms are commonly used. Several researchers have been successful in using variations of these algorithms to generate OAs from symmetric ILPs (see [2], [3], [8], and [7]).

For symmetric ILPs, many of the subproblems in the ILP enumeration tree are isomorphic, meaning that a substantial amount of computational effort is wasted on solving subproblems that are identical to each other [23]. Hence for symmetric ILPs with large symmetry groups, it can be very difficult to find optimal solutions or prove infeasibility

using the classical algorithms. Margot [22] developed a solver that eliminates evaluation of isomorphic subproblems in the ILP enumeration tree, thereby greatly speeding up the enumeration of the ILP solutions. The Margot ILP solver [22] exploits the symmetry group of an ILP in pruning its enumeration tree.

Bulutoglu and Margot [7] defined the *isomorphism group* $G_{s,k}$ of a k factor, s-level design $\mathbf{D}$ to be the group of all permutations of factors as well as permutations of levels within each factor in $\mathbf{D}$. The definition of $G_{s,k}$ does not mention runs. This is because run order is irrelevant when a factorial design is viewed as a set of frequencies for the set of distinct runs possible in the design [7]. There exists an element $g \in G_{s,k}$ such that $g(\mathbf{x}_1) = \mathbf{x}_2$ if and only if $\mathbf{x}_1$ and $\mathbf{x}_2$ are the indicator vectors of isomorphic designs $\mathbf{D}_1$ and $\mathbf{D}_2$. Note that $\left|G_{s,k}\right| = k!(s!)^k$ and $G_{s,k} \cong S_s \wr S_k$, where $\wr$ is the wreath product. Bulutoglu and Margot [7] showed that the Branch-and-Cut with isomorphism pruning algorithm in Margot [22] returns exactly one OA(N, k, s, t) for each isomorphism class if it uses the group $G_{s,k}$ [7].

The *automorphism group* of an ILP of the form

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{d} \geq \mathbf{x} \geq \mathbf{0}, \quad d_i, x_i \in \mathbb{Z}_{\geq 0}, \end{aligned} \tag{2.14}$$

is defined as

$$G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = \{\pi \in S_n \mid \pi(\mathbf{c}) = \mathbf{c}, \pi(\mathbf{d}) = \mathbf{d} \text{ and } \exists\, \sigma \in S_m : \mathbf{A}(\pi, \sigma) = \mathbf{A} \text{ and } \sigma(\mathbf{b}) = \mathbf{b}\}, \tag{2.15}$$

where $\mathbf{A}(\pi, \sigma)$ is the matrix obtained by permuting the rows of $\mathbf{A}$ according to σ and columns according to π. Bulutoglu and Margot [7] proved the following theorem:

Theorem 3. *For ILP (2.4), let $G(\boldsymbol{A}, \lambda\mathbf{1}, \mathbf{1}, p_{\max}\mathbf{1})$ be the automorphism group of the ILP, $G_{s,k}$ be the isomorphism group of an OA(N, k, s, t) and S_{s^k} be the set of all possible permutations of $\left\{1, 2, ..., s^k\right\}$. Then, $G_{s,k} \subseteq G(\boldsymbol{A}, \lambda\mathbf{1}, \mathbf{1}, p_{\max}\mathbf{1})$ for $0 < t < k$ and $G_{s,k} \subset G(\boldsymbol{A}, \lambda\mathbf{1}, \mathbf{1}, p_{\max}\mathbf{1}) = S_{s^k}$ otherwise.*

Margot [23] declares two solutions to ILP (2.14) to be isomorphic if one can be obtained from the other by applying a permutation belonging to the automorphism group $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$. Bulutoglu and Margot [7] used the isomorphism pruning integer programming solver developed by Margot [22] to enumerate all non-isomorphic orthogonal arrays for many N, k, s, t combinations. Their enumeration was based on finding all non-isomorphic solutions to ILP (2.4) using the group $G(\mathbf{A}, \lambda\mathbf{1}, \mathbf{1}, p_{\max}\mathbf{1})$. This was allowed, as in all the cases they considered, $\left|G_{s,k}\right| = |G(\mathbf{A}, \lambda\mathbf{1}, \mathbf{1}, p_{\max}\mathbf{1})|$.

The results obtained by Bulutoglu and Margot [7] based on Theorem 3 indicate that there are major efficiencies to be gained by using the isomorphism pruning method in Margot [22] to solve ILPs of the form in ILPs (2.4) and (2.13). However, in order for the Margot ILP solver [22] to be more effective, an efficient methodology for identifying all symmetries in ILPs is required.

2.5 Identifying Symmetries in ILPs

Clearly, the automorphism group of an ILP of the form (2.14) is a subgroup of its symmetry group. However, there is no guarantee that all symmetries of such an ILP will be captured by its automorphism group. In fact, it is easy to construct examples of ILPs with equality constraints only in which the symmetry group is much larger than the automorphism group. On the other hand, identifying the full symmetry group G for an ILP formulation is a difficult open problem.

Therefore, it is worthwhile to develop techniques to find a set of generators of the symmetry group for any ILP. The most difficult obstacle to overcome when developing such a technique is that the makeup of G is inherently determined by the ILP's feasible set [23]. For example, the definition of a symmetry group says that $G = S_n$ for an infeasible ILP with n variables. Margot [23] proved that deciding if $G = S_n$ is an *NP-Complete* problem. Hence, finding the symmetry group G of an ILP is *NP-Hard*. Assuming $P \neq NP$, there is no way to find a polynomial-time algorithm for finding generators of G. One approach that

is often effective is to find the symmetry group of the LP relaxation of the ILP. Margot [23] defined this group as the set of all permutations of the LP that send feasible points to feasible points with the same objective function value. G^{LP} is the set of all permutations that send facets of the LP relaxation to facets and fixes the objective function provided that the LP relaxation is full dimensional. Then for an ILP in the form

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x}, \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b}, \\ & \mathbf{x} \in \mathbb{Z}^n_{\geq 0}, \end{aligned} \tag{2.16}$$

$G(\mathbf{A}, \mathbf{b}, \mathbf{c}) = G^{\text{LP}}$ provided that the LP relaxation of the ILP has no redundant constraints and contains interior points (*i.e.* it is full dimensional). On the other hand, for an ILP of the form (2.14), all we can say is $G(\mathbf{A}, \mathbf{b}, \mathbf{c}) \leq G^{\text{LP}}$. In Chapter 4, methods are developed to find G^{LP} for ILPs with equality constraints only. G^{LP}s of LPs with equality constraints have not yet been studied in the literature. $\mathcal{F}(k, s, t) \subseteq \mathcal{F}(k, s, t)^{\text{LP}}$ implies $G^{\text{LP}} \leq G$ as non-integer solutions in $\mathcal{F}(k, s, t)^{\text{LP}}$ can break symmetries that would otherwise be present in G [23]. It is possible that working with G^{LP} may be less efficient, but for most ILP formulations of OAs are such that G^{LP} is either a large subgroup of G or it is equal to G itself.

Now, consider the matrix

$$\mathbf{L} = \begin{bmatrix} \mathbf{0} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{A}$ is the $(0, 1)$ ILP (2.16) constraint matrix. The matrix $\mathbf{L}$ can be seen as the adjacency matrix of a bipartite graph K with colored vertices. K has one vertex for each row and one vertex for each column of $\mathbf{A}$. The set of vertices for the rows constitute one side of the bipartition and the vertices for the columns constitute the other. Two column (row) vertices in K have the same color if and only if their corresponding objective function (right hand side) coefficients are equal. Margot [23] showed that the automorphism group of K is equal to $G(\mathbf{A}, \mathbf{b}, \mathbf{c})$. (The automorphism group of a vertex colored graph is the set of

all permutations of its vertices that send adjacent vertices to adjacent vertices and preserve colors.)

In cases where the ILP constraint matrix **A** is not (0, 1), there are many mappings of the ILP to an edge colored vertex colored graph such that the color preserving automorphisms of the graph correspond to the symmetries of the ILP. One example in the literature is in [30]. In this work, the code created by Bulutoglu and Ryan [8] is used for this purpose. The problem of determining the automorphism group of a graph is at least as computationally complex as the famous Graph Isomorphism problem [23]. However, there are software packages available such as Nauty [25] and GAP [12] that can solve this problem in most instances.

2.6 Margot ILP Solver [22]

The Margot ILP solver [22] utilizes a modified version of the Branch-and-Cut algorithm for solving ILPs. The modification is in the pruning step. Instead of pruning just one subproblem, the Margot ILP solver [22] prunes the entire set of isomorphic subproblems. For ILPs whose symmetry groups are large, this modification greatly increases the efficiency of the Branch-and-Cut if one is able to exploit a large subgroup of the symmetry group.

Margot developed his modified Branch-and-Cut algorithm over a series of papers [19], [20], [21] and [22]. The reader is referred to [21] for the most basic version of his Branch-and-Cut algorithm for binary ILPs. In [19], he made his algorithm more efficient by introducing zero setting. In [20], he incorporated strict zero setting and ranked branching. In [22], he generalized his algorithm in [20] to bounded ILPs where each variable is bounded by a constant.

The next chapter discusses theoretical work that lead to the research objectives and results listed in Chapter 4.

III. Theoretical Research

3.1 Chapter Introduction

This chapter dicusses the theoretical research efforts based on the review of current literature provided in Chapter 2. In Section 3.2, Lemma 1 is used to define the linear relaxation orthogonal array polytope of an OA($\lambda s^t, k, s, t$). An OA($\lambda s^t, k, s, t$) exists if and only if its linear relaxation orthogonal array polytope contains integer vectors where each integer vector represents an OA($\lambda s^t, k, s, t$). In this section, it is shown that the stated formulation defining the linear relaxation orthogonal array polytope does not contain redundant inequalities. This implies that each inequality is a facet. In Section 3.3, a large subgroup of all the permutation symmetries of the linear relaxation orthogonal array polytope of OA($\lambda s^t, k, s, t$) is found. It is shown that the subgroup found is equal to the group of all the permutation symmetries. This result is computationally verified for many cases.

3.2 The Linear Relaxation OA Polytope

The following theorem follows immediately from Lemma 1 in Section 3.2 by taking $\mathbf{z} = \mathbf{1}$ and observing that $N_{\mathbf{x}} \geq 0$.

Theorem 4. *Let* $\{a_c\}$ *and* $I_{\mathbf{x}}$ *be as in Lemma 1. Let* $0 \leq d(\mathbf{1}, \mathbf{x}) \leq t$ *and*

$$\begin{aligned} a_{t-d(\mathbf{1},\mathbf{x})} + (-1)^{t-d(\mathbf{1},\mathbf{x})+1} \sum_{\substack{\mathbf{y} \in J_{\mathbf{x}} \\ d(\mathbf{1},\mathbf{y}) > t}} \binom{d(\mathbf{1},\mathbf{y}) - d(\mathbf{1},\mathbf{x}) - 1}{t - d(\mathbf{1},\mathbf{x})} N_{\mathbf{y}} \geq 0, \\ N_{\mathbf{y}} \geq 0, \end{aligned} \tag{3.1}$$

where $J_{\mathbf{x}} = \{\mathbf{y} \in \{1, \cdots, s\}^k : y_i = x_i \ \ \forall i \in I_{\mathbf{x}}\}$. *Then an OA*($\lambda s^t, k, s, t$) *exists if and only if there exist integers* $N_{\mathbf{y}}$ *satisfying the inequalities (3.1).*

Theorem 4 converts the Bulutoglu and Margot [7] ILP with equalities to an ILP problem with inequalities only. This is done by deleting the set of basic variables $N_{\mathbf{x}}$ with

$d(\mathbf{x}, \mathbf{1}) \leq t$ after implementing Gaussian elimination. Deletion of these variables is only possible because the coefficients of all $N_{\mathbf{x}}$ are 1 and the coefficients of $N_{\mathbf{y}}$ with $d(\mathbf{y}, \mathbf{1}) > t$ are all integers at the end of Gaussian elimination. The substance of Lemma 1 from [29] is that it correctly identifies which set of basic variables has elements that are all integer combinations of the remaining variables. This enables the deletion of this set of variables from the Bulutoglu and Margot [7] ILP. The new ILP has the form

$$\begin{aligned} \min \quad & 0 \\ \text{s.t.} \quad & \mathbf{A}'(k, s, t)\mathbf{x}' \geq \mathbf{b}_m \\ & \mathbf{x} \in \{0, 1, 2, ..., p_{\max}\}^{s^k - m}, \end{aligned} \tag{3.2}$$

where $m = \sum_{i=0}^{t} \binom{k}{i} (s-1)^i$, and $\mathbf{A}'(k, s, t)$, $\mathbf{b}_m$, $\mathbf{x}'$ are the results of applying Gaussian elimination to $\left[\mathbf{A}(k, s, t) \mid \lambda \mathbf{1}_{\binom{k}{t}s^t}\right]$ and deleting the basic variables $N_{\mathbf{x}}$.

For a general ILP with only equality constraints over non-negative integer vectors, deleting variables by using Gaussian elimination may not always be possible. This can be the case for one of two reasons. First, there may be no set of basic variables whose elements are integer combinations of the remaining variables plus some integer. Second, it may be very difficult to identify such a set of basic variables. For example, for the ILP in [7] when $k = 8$, $s = 2$ and $t = 3$, the proportion of such sets of of basic variables to all sets of basic variables was estimated to be .5%. This estimate was calculated by repeating the following procedure 1000 times.

1. Randomly permute the columns of the constraint matrix.
2. Augment the resulting matrix with its right hand side.
3. Row reduce it to its reduced row echelon form.
4. Record if the output has only integer entries.

The MATLAB code to execute this procedure is given in Section A.2.

The following definition arises naturally from Theorem 4.

Definition 3. *Let* $m = \sum_{i=(t+1)}^{k} \binom{k}{i}(s-1)^i$. *The set of* $\mathbf{N}^y \in \mathbb{R}^m$ *with* $k \geq d(\mathbf{1}, \mathbf{y}) \geq t+1$ *that satisfy the system of inequalities (3.1) is called the* linear relaxation polytope *of* $OA(\lambda s^t, k, s, t)$.

Let $\text{OAP}(k, s, t, \lambda)$ denote the linear relaxation polytope of $\text{OA}(\lambda s^t, k, s, t)$. Note that $\text{OAP}(k, s, t, \lambda)$ is a polytope not an unbounded polyhedron because it can be embedded inside the hypercube $[0, \lambda]^m$.

Theorem 5. *The* $OAP(k, s, t, \lambda)$ *is full dimensional for all* λ.

Proof. Let m be as above and $\mathbf{1}_m$ be the $m \times 1$ vector of all 1s. This result is proven by showing that $\lambda/s^{k-t}\mathbf{1}_m$ is an interior point of the $\text{OAP}(k, s, t, \lambda)$. Rosenberg [29] showed that the system of equations in Lemma 1 is equivalent to

$$\sum_{\substack{x \in [s]^k \\ x_i = a_i \forall i \in I}} N_{\mathbf{x}} = \lambda \quad \text{for all } t\text{-subsets } I \text{ of } [k] \text{ and all } \mathbf{a} \in [s]^k \tag{3.3}$$

where $[s]^k = \{1, 2, \cdots, s\}^k$ and $[k] = \{1, 2, \cdots, k\}$. It is clear that $\lambda/s^{k-t}\mathbf{1}_{s^k}$ solves the system of equations (3.3) in $\mathbb{R}^{s^k}_{\geq 0}$. Then $N_{\mathbf{x}} = \lambda/s^{k-t}$ and $N_{\mathbf{y}} = \lambda/s^{k-t}$ for $d(\mathbf{x}, \mathbf{1}) \leq t$ and $d(\mathbf{y}, \mathbf{1}) > t$ respectively solves the system of equations (2.2) Lemma 1 in $\mathbb{R}^{s^k}_{\geq 0}$. Now, clearly $\lambda/s^{k-t}\mathbf{1}_m$ satisfies all the inequalities defining the $\text{OAP}(k, s, t, \lambda)$ strictly. Hence, it is an interior point. □

The following theorem proves that no constraint in Theorem 4 is redundant unless $k = t+1$ and $s = 2$.

Theorem 6. *Each one of the distinct* s^k *inequalities in Theorem 4 defining the* $OAP(k, s, t, \lambda)$ *is a facet and no facet is repeated unless* $k = t+1$ *and* $s = 2$.

Proof. At least one of the inequalities in (3.1) is a facet since otherwise $\text{OAP}(k, s, t, \lambda)$ would be an unbounded polyhedron. Then there exists $\mathbf{N}^y \in \mathbb{R}^m$ satisfying all but the

facet defining inequality in (3.1). Let $N_{\mathbf{u}_0}^{f(\mathbf{u}_0)} < 0$, $N_{\mathbf{u}_0}^{f(\mathbf{u}_0)} \in \mathbb{R}$ be the left hand side in constraints (2.2) corresponding to the facet defining inequality in (3.1), where

$$f(\mathbf{u}) = \begin{cases} x & \text{if } d(\mathbf{u}, \mathbf{1}) \leq t, \\ y & \text{otherwise.} \end{cases}$$

Hence, there exists vectors $\mathbf{N}^y \in \mathbb{R}^m$ and $\mathbf{N}^x \in \mathbb{R}^{s^k-m}$ that satisfy the equations in constraints (2.2) such that $N_{\mathbf{y}}^y \geq 0$ and $N_{\mathbf{x}}^x \geq 0$ for $\mathbf{x} \neq \mathbf{u}_0$ and $\mathbf{y} \neq \mathbf{u}_0$, where $N_{\mathbf{u}_0}^{f(\mathbf{u}_0)} < 0$. The group $G_{s,k} \cong S_s \wr S_k$ sends vectors in $\mathbb{R}^{s^k}$ that satisfy the equations in constraints (2.2) to vectors that satisfy the same equations. Furthermore $G_{s,k}$ acts transitively on the variables of constraints (2.2). Hence, for each $\mathbf{u}_0 \in [s]^k$, there exists a solution with $N_{\mathbf{u}_0}^{f(\mathbf{u}_0)} < 0$ and $N_{\mathbf{w}}^{f(\mathbf{w})} \geq 0$ for $\mathbf{w} \neq \mathbf{u}_0$, $\mathbf{w} \in [s]^k$. Then there exists $\mathbf{N}^y \in \mathbb{R}^m$ satisfying all but one facet defining inequality in (3.1) whose left hand side is $N_{\mathbf{u}_0}^{f(\mathbf{u}_0)} < 0$ in constraints (2.2) for arbitrary $\mathbf{u}_0 \in [s]^k$. Hence, there are no distinct redundant inequalities in (3.1) and each distinct inequality is a facet.

Observe that unless $k = t + 1$ and $s = 2$, $\{\mathbf{y} \in J_{\mathbf{x}_1} : d(\mathbf{1}, \mathbf{y}) > t\} \neq \{\mathbf{y} \in J_{\mathbf{x}_2} : d(\mathbf{1}, \mathbf{y}) > t\}$ whenever $\mathbf{x}_1 \neq \mathbf{x}_2$. Hence, no facet is repeated unless $k = t+1$ and $s = 2$. For the degenerate case $k = t + 1$ and $s = 2$, there is only one variable $N_{(2,2,\cdots,2)}^y$ and one gets $N_{(2,2,\cdots,2)}^y \geq 0$ and $-N_{(2,2,\cdots,2)}^y \geq -\lambda$ each repeated 2^{k-1} times. □

Remark 1. *While it is true that $S_s \wr S_k$ acts as a group of symmetries on OAP(k, s, t, λ), this action is no longer as a group of linear transformations (as is the case for the full system of equations (3.3)), but rather as a group of affine transformations. In particular $S_s \wr S_k$ does not permute the variables of OAP(k, s, t, λ), but rather, it acts by permuting the half-spaces defined by inequalities (3.1). Furthermore, this action is transitive.*

Theorem 6 was verified when $\lambda = 1$ for each of $s = 2$, $4 \leq k \leq 13$, $2 \leq t \leq k - 2$, $s = 3$, $3 \leq k \leq 8$, $2 \leq t \leq k - 1$ and $s = 4$, $3 \leq k \leq 6$, $2 \leq t \leq k - 1$ cases. This verification was based on finding interior points on each facet of the OAP(k, s, t, λ).

Let $\mathbf{Bx} \leq \mathbf{d}$ be the system of inequalities in Theorem 4 defining the OAP(k, s, t, λ). Let $\mathbf{B}^i\mathbf{x} \leq \mathbf{d}^i$ be the same system after the i'th inequality is deleted. Also, let $(\mathbf{b}_i)^T$ be the i'th row of $\mathbf{B}$ and F_i be the hyperplane defined by the equality $(\mathbf{b}_i)^T\mathbf{x} = d_i$. To find interior points on $F_i \cap \text{OAP}(k, s, t, \lambda)$, each face of the OAP$(k, s, t, \lambda)$, feasible solutions were found to the following linear program

$$\begin{aligned} \min \quad & \mathbf{1}^T\mathbf{x} \\ \text{such that} \quad & (\mathbf{b}^i)^T\mathbf{x} = d_i \\ & \mathbf{B}^i\mathbf{x} \leq \mathbf{d}^i - \tfrac{1}{1000}\mathbf{1}. \end{aligned}$$

The MATLAB code to implement this test is found in Section A.4.

3.3 Permutation Symmetries of the OA Polytope

First, a definition for the permutation symmetries of a polytope:

Definition 4. *Let P be a full dimensional polytope in $\mathbb{R}^m$. A permutation of coordinates of $\mathbb{R}^m$ that also sends P onto itself is called a* permutation symmetry of P. The set of all such transformations forms a group called the *permutation symmetry group* ($\Pi(P)$) of P.

The variables $N_{\mathbf{y}}$ in Theorem 4 are indexed by all factor level combinations $\mathbf{y} \in [s]^k$ with $d(\mathbf{y}, \mathbf{1}) > t$ and $\Pi(\text{OAP}(k, s, t, \lambda))$ permutes those variables. The following theorem explicitly describes a nontrivial subgroup of $\Pi(\text{OAP}(k, s, t, \lambda))$.

Theorem 7. *Let*

$$H_{k,s,t} \cong \begin{cases} S_{s^k-1} & \text{if } t = 0, \\ S_{(s-1)} \wr S_k & \text{if } 0 < t < k \text{ and } (k > t+1 \text{ or } s > 2), \\ I & \text{otherwise,} \end{cases}$$

and I be the identity group. Also, let $Y = \{\mathbf{y} \in [s]^k : d(\mathbf{y}, \mathbf{1}) > t\}$. Then, when $H_{k,s,t}$ is not defined to be I, it naturally embeds as the group of permutations that preserve Y, and $H_{k,s,t} \subseteq \Pi(OAP(k, s, t, \lambda))$.

Proof. It is easy to see that each element of $H_{k,s,t}$ maps Definition 3 defining constraints of the OAP(k, s, t, λ) to each other. Then, $\mathbf{v} \in \text{OAP}(k, s, t, \lambda) \Rightarrow h(\mathbf{v}) \in \text{OAP}(k, s, t, \lambda)$ for all $h \in H_{k,s,t}$. Hence, $H_{k,s,t}$ maps the OAP(k, s, t, λ) into itself. On the other hand, for a given $\mathbf{v} \in \text{OAP}(k, s, t, \lambda)$, one has $h(h^{-1}(\mathbf{v})) = \mathbf{v}$ as $h^{-1}(\mathbf{v}) \in \text{OAP}(k, s, t, \lambda)$. This implies that $h(\text{OAP}(k, s, t, \lambda)) = \text{OAP}(k, s, t, \lambda)$ for each h. Hence, $H_{k,s,t} \subseteq \Pi(\text{OAP}(k, s, t, \lambda))$. The second part of the theorem follows directly from the definition of wreath product of groups. □

Remark 2. *For $k = t + 1$ and $s = 2$ there is only one variable $N_{\mathbf{y}}$ with $d(\mathbf{y}, \mathbf{1}) > t$, hence $\Pi(OAP(k, s, t, \lambda)) = I$. For the case $t = 0$, there is one constraint on the $s^k - 1$ variables. The coefficients of this constraint are all -1s, hence $H_{k,s,0} \cong S_{s^k-1}$. If $t = k$ there are no variables in inequalities (3.1).*

Next, tools are developed for computing $\Pi(\text{OAP}(k, s, t, \lambda))$. As noted in [23], the set of all permutations of coordinates in $\mathbb{R}^m$ mapping P onto itself consists of all permutations of coordinates that map facets of P onto its facets. Hence, if all the facets of an OAP(k, s, t, λ) are known, then $\Pi(\text{OAP}(k, s, t, \lambda))$ can be computed explicitly. $\Pi(\text{OAP}(k, s, t, \lambda))$ were computed explicitly for all the k, s, t combinations in which Theorem 6 was verified. This was done by first finding

$$G_{k,s,t} = \{\pi \mid \exists\, \sigma : \mathbf{A}_{k,s,t}(\pi, \sigma) = \mathbf{A}_{k,s,t}\}$$

where $\mathbf{A}_{k,s,t}$ is the constraint matrix of inequalities (3.1) in Theorem 4 and $\mathbf{A}_{k,s,t}(\pi, \sigma)$ is the resulting matrix when the rows of $\mathbf{A}_{k,s,t}$ are permuted according to σ and columns according to π. Then, $H_{k,s,t} \leq \Pi(\text{OAP}(k, s, t, \lambda)) \leq G_{k,s,t}$ as $\Pi(\text{OAP}(k, s, t, \lambda))$ must preserve the constraint matrix. Calculating $G_{k,s,t}$ over directly calculating $\Pi(\text{OAP}(k, s, t, \lambda))$ was done for the sake of convenience. Finding only $G_{k,s,t}$ proved to be sufficient in all the cases

considered. $G_{k,s,t}$ was calculated as described in [23], by first mapping the matrix

$$\begin{bmatrix} \mathbf{0} & \mathbf{A}_{k,s,t} \\ \mathbf{A}_{k,s,t}^T & \mathbf{0} \end{bmatrix}$$

to an edge colored graph and then finding the automorphism group. The code to generate the constraints (3.1) is found in Section A.3. The code used to generate $G_{k,s,t}$ for various values of k, s and t can be found in Section A.4. Nauty software [25] was used to find the automorphism groups. In all the k, s, t cases studied, it was found that $|G_{k,s,t}| = ((s-1)!)^k k! = |H_{k,s,t}|$ implying $H_{k,s,t} = \Pi(\text{OAP}(k, s, t, \lambda))$. To prove this observation, the following two lemmas are required.

Lemma 3. *Let GG be the group of maps ϕ from $[s]^k$ to $[s]^k$ that preserve the Hamming distance, i.e. $d(\mathbf{x}, \mathbf{y}) = d(\phi(\mathbf{x}), \phi(\mathbf{y}))$ for all $\phi \in GG$ then $GG \cong S_s \wr S_k$. Furthermore, if G_1 is the subgroup of GG such that $\tau(\mathbf{1}) = \mathbf{1}$ for all τ in G_1 then $G_1 \cong S_{(s-1)} \wr S_k$.*

Proof. Replace $\mathbb{F}_q$ with $\mathbb{Z}_s$, the ring of integers mod s and $\mathbb{F}_q^*$ with $\mathbb{Z}_s - \{0\}$ in Theorem 5 and Theorem 6 as well as in their proofs in [11]. Also, replace the term "vector space" with "$\mathbb{Z}_s^n$". Then the resulting theorems and their proofs are still valid as the proofs never use the multiplicative invertibility of non-zero elements. Now, replace $\{0, 1, \cdots, s-1\}$ with $\{1, \cdots, s\}$ to get

$$GG = S_{\{1,\cdots,s\}} \wr S_k \cong S_s \wr S_k$$

and

$$G_1 = S_{\{2,\cdots,s\}} \wr S_k \cong S_{(s-1)} \wr S_k.$$

□

Lemma 4. *If $k - t \geq 2$ then $\{a_c\}_{c=0}^t$ in Theorem 4 are all non-zero and distinct.*

Proof. $\{a_c\}_{c=0}^t$ in Lemma 1 are the same $\{a_c\}_{c=0}^t$ in Theorem 4. $N_{\mathbf{x}} = \lambda/s^{k-t}$ and $N_{\mathbf{y}} = \lambda/s^{k-t}$ solves the Lemma 1 system of constraints. Plugging in $N_{\mathbf{x}} = \lambda/s^{k-t}$, $N_{\mathbf{y}} = \lambda/s^{k-t}$ and

multiplying both sides of equations (2.2) by s^{k-t}/λ leads to

$$\frac{s^{k-t}}{\lambda}a_{t-d(\mathbf{z},\mathbf{x})} = 1 + (-1)^{t-d(\mathbf{z},\mathbf{x})} \sum_{\substack{\mathbf{y}\in J_{\mathbf{x}} \\ d(\mathbf{z},\mathbf{y})>t}} \binom{d(\mathbf{z},\mathbf{y}) - d(\mathbf{z},\mathbf{x}) - 1}{t - d(\mathbf{z},\mathbf{x})}. \tag{3.4}$$

Taking $\mathbf{z} = \mathbf{1}$ and $\mathbf{x} = (2(\mathbf{1}_r)^T, (\mathbf{1}_{k-r})^T)$ equation (3.4) implies that

$$\frac{s^{k-t}}{\lambda}a_{t-r} = 1 + (-1)^{t-r} \sum_{i=1}^{k-t} (s-1)^{(t-r+i)} \binom{k-r}{t+i-r} \binom{t+i-r-1}{t-r}. \tag{3.5}$$

There are $k - t \geq 2$ positive integers inside the summation in equation (3.5). This implies that $a_c \neq 0$ for $0 \leq c \leq t$ and $a_c a_{c+1} < 0 \Rightarrow a_c \neq a_{c+1}$ for $0 \leq c \leq t-1$. Furthermore, if $a_{t-r_1} = a_{t-r_2}$ for some $1 \leq r_2 < r_1 \leq t$, then one must have $r_1 \equiv r_2 \pmod 2$. This further implies that

$$\sum_{i=1}^{k-t} (s-1)^{(t-r_1+i)} \binom{k-r_1}{t+i-r_1} \binom{t+i-r_1-1}{t-r_1} = \sum_{i=1}^{k-t} (s-1)^{(t-r_2+i)} \binom{k-r_2}{t+i-r_2} \binom{t+i-r_2-1}{t-r_2}$$

for some $0 \leq r_2 < r_1 \leq t$. However, this is impossible as

$$0 < (s-1)^{(t-r_1+i)} \binom{k-r_1}{t+i-r_1} \binom{t+i-r_1-1}{t-r_1} < (s-1)^{(t-r_2+i)} \binom{k-r_2}{t+i-r_2} \binom{t+i-r_2-1}{t-r_2}.$$

Hence, $a_{t-r_1} \neq a_{t-r_2}$ and $|a_{t-r_1}| < |a_{t-r_2}|$ for all $1 \leq r_2 < r_1 \leq t$. Finally, each a_c is divisible by $\lambda = a_0$ by the nature of the difference equation defining a_c and $|a_c|$ is strictly increasing with c as c goes from 1 to t. Hence, $a_c \neq a_0 = \lambda$ for $c \neq 0$. □

Equation (3.5) provides a closed form formula for the solution of the inhomogeneous recurrence relation of degree $k-t+1$ in Lemma 1. This recurrence relation is equivalent to a homogeneous recurrence relation of degree $k - t + 2$. Solving such an equation requires finding all complex roots of a degree $k-t+2$ polynomial. Coming up with this closed form formula for arbitrary values of $k - t$ without relating a_c to the orthogonal array problem appears to be difficult.

Theorem 8. $\Pi(OAP(k, s, t, \lambda)) = H_{k,s,t}$.

Proof. By Remark 2 it suffices to consider the case $1 \le t \le k-1$ and ($k > t+1$ or $s > 2$). For $\mathbf{z} = \mathbf{1}$, let GR be the group of all coordinate permutations of $[s]^k$ which permute the facets in Theorem 4. Then,

$$\mathbf{y} \in J_{\mathbf{x}} \Leftrightarrow \sigma(\mathbf{y}) \in J_{\sigma(\mathbf{x})} \tag{3.6}$$

and $a_{t-d(\mathbf{1},\mathbf{x})} = a_{t-d(\sigma(\mathbf{1}),\sigma(\mathbf{x}))}$ for all $\mathbf{x}$ such that $0 \le d(\mathbf{1},\mathbf{x}) \le t$ and all $\sigma \in GR$. The distinctness of $\{a_c\}_{c=0}^{t}$ further implies that

$$d(\mathbf{1},\mathbf{x}) = d(\sigma(\mathbf{1}),\sigma(\mathbf{x})) \quad \text{if} \quad d(\mathbf{1},\mathbf{x}) \le t.$$

Since

$$a_{t-d(\mathbf{1},\mathbf{1})} + (-1)^{t-d(\mathbf{1},\mathbf{1})+1} \sum_{\substack{\mathbf{y}\in J_{\mathbf{1}} \\ d(\mathbf{1},\mathbf{y})>t}} \binom{d(\mathbf{1},\mathbf{y}) - d(\mathbf{1},\mathbf{1}) - 1}{t - d(\mathbf{1},\mathbf{1})} N_{\mathbf{y}} \ge 0,$$

when $\mathbf{x} = \mathbf{1}$, is the only inequality in inequalities (3.1) that has a_t as its constant term, it must be mapped to itself by GR. Then, GR must preserve $\mathbf{1}$, otherwise $J_{\mathbf{1}} \ne J_{\sigma(\mathbf{1})}$ as $k > t+1$ or $s > 2$. Furthermore, GR must also preserve the set $\{\mathbf{y} : d(\mathbf{1},\mathbf{y}) > t\}$. Let GR' be the subgroup of GR consisting of all permutations that only permute the variables of the facet defining inequalities in Theorem 4 without changing their coefficients. Then GR' must preserve the coefficients $\binom{d(\mathbf{1},\mathbf{y})-d(\mathbf{1},\mathbf{1})-1}{t-d(\mathbf{1},\mathbf{1})}$ for $\{\mathbf{y} : d(\mathbf{1},\mathbf{y}) > t\}$. Hence,

$$d(\mathbf{1},\mathbf{y}) = d(\tau(\mathbf{1}),\tau(\mathbf{y})) = d(\mathbf{1},\tau(\mathbf{y})) \quad \text{if} \quad d(\mathbf{1},\mathbf{y}) > t$$

for all $\tau \in GR'$. Since $GR' \le GR$, this leads to

$$d(\mathbf{1},\mathbf{y}) = d(\tau(\mathbf{1}),\tau(\mathbf{y})) = d(\mathbf{1},\tau(\mathbf{y})) \quad \text{if} \quad d(\mathbf{1},\mathbf{y}) \le t.$$

This shows that all elements in GR' are rotations around $\mathbf{1}$.

Let $T = \{\mathbf{u} \in [s]^k : d(\mathbf{u},\mathbf{1}) = 1\}$. Then $|T| = (s-1)k$ and $|J_{\mathbf{u}}| = s^{k-1}$ for each $\mathbf{u} \in T$. Since each $\tau \in GR'$ is an invertible rotation around $\mathbf{1}$,

$$\tau(T) = \{\tau(\mathbf{u}) : \mathbf{u} \in T\} = T \quad \text{for all } \tau \in GR',$$

and consequently

$$\{J_{\mathbf{u}} : \mathbf{u} \in T\} = \{J_{\tau(\mathbf{u})} : \mathbf{u} \in T\} \quad \text{for all } \tau \in GR'. \tag{3.7}$$

For $\mathbf{x}, \mathbf{y} \in [s]^k$, let $r_1(\mathbf{x}, \mathbf{y})$, $r_2(\mathbf{x}, \mathbf{y})$, $r_3(\mathbf{x}, \mathbf{y})$, and $r_4(\mathbf{x}, \mathbf{y})$ be the number of $\mathbf{u} \in T$ such that $\mathbf{x}, \mathbf{y} \in J_{\mathbf{u}}$, $\mathbf{x}, \mathbf{y} \notin J_{\mathbf{u}}$, $\mathbf{x} \in J_{\mathbf{u}}, \mathbf{y} \notin J_{\mathbf{u}}$ and $\mathbf{x} \notin J_{\mathbf{u}}, \mathbf{y} \in J_{\mathbf{u}}$ respectively. Let

$$\mathbf{W} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$$

and $a_1, a_2, a_3, a_4, a_5 \in \{2, \cdots, s\}$ such that $a_4 \neq a_5$. Now, let $\alpha_1(\mathbf{x}, \mathbf{y}), \alpha_2(\mathbf{x}, \mathbf{y}), \alpha_3(\mathbf{x}, \mathbf{y})$, $\alpha_4(\mathbf{x}, \mathbf{y}), \alpha_5(\mathbf{x}, \mathbf{y})$ be the number of columns in $\mathbf{W}$ of the form

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ a_1 \end{bmatrix}, \begin{bmatrix} a_2 \\ 1 \end{bmatrix}, \begin{bmatrix} a_3 \\ a_3 \end{bmatrix}, \begin{bmatrix} a_4 \\ a_5 \end{bmatrix}$$

respectively. Then,

$$\begin{aligned} r_1(\mathbf{x}, \mathbf{y}) &= \alpha_4(\mathbf{x}, \mathbf{y}), \\ r_2(\mathbf{x}, \mathbf{y}) &= k(s-1) - (r_1(\mathbf{x}, \mathbf{y}) + r_3(\mathbf{x}, \mathbf{y}) + r_4(\mathbf{x}, \mathbf{y})), \\ r_3(\mathbf{x}, \mathbf{y}) &= \alpha_3(\mathbf{x}, \mathbf{y}) + \alpha_5(\mathbf{x}, \mathbf{y}), \\ r_4(\mathbf{x}, \mathbf{y}) &= \alpha_2(\mathbf{x}, \mathbf{y}) + \alpha_5(\mathbf{x}, \mathbf{y}). \end{aligned} \tag{3.8}$$

By (3.6) and (3.7), one can see that

$$r_i(\mathbf{x}, \mathbf{y}) = r_i(\tau(\mathbf{x}), \tau(\mathbf{y})), \tag{3.9}$$

for all $\tau \in GR'$ and $i = 1, 2, 3, 4$.

Let $R_i(\mathbf{x}')$ be the set of all $\mathbf{x}$ obtained from $\mathbf{x}' \in \{2, \cdots, s\}^k$ by replacing exactly i coordinates in $\mathbf{x}'$ with 1. Let $B(\mathbf{x}') = \cup_{i=0}^{k} R_i(\mathbf{x}')$ and $\mathbf{B}(\mathbf{x}')$ be the matrix whose $(\sum_{j=0}^{i-1} \binom{k}{j} + 1)$th row to $(\sum_{j=0}^{i} \binom{k}{j})$th row be the elements of $R_i(\mathbf{x}')$ in some order. Then, the multiset

$$\biguplus_{\mathbf{x}' \in \{2, \cdots, s\}^k} B(\mathbf{x}')$$

covers every element in $[s]^k$ at least once, and each element of $R_i(\mathbf{x}')$ is covered exactly $(s-1)^i$ times. Also, $|B(\mathbf{x}')| = 2^k$ for each $\mathbf{x}' \in \{2, \cdots, s\}^k$ and there are $(s-1)^k$ such $B(\mathbf{x}')$. Furthermore, $|B(\mathbf{x}') \cap B(\mathbf{y}')| = 2^{(k-d(\mathbf{x}',\mathbf{y}'))}$, the ith column of $\mathbf{B}(\mathbf{x}')$ has only 1 and $\mathbf{x}'[i]$ as its entries and consequently for any pair $\mathbf{x}, \mathbf{y} \in B(\mathbf{x}')$, $\alpha_5(\mathbf{x}, \mathbf{y}) = 0$. Then, equations (3.8) and (3.9) imply that $\alpha_i(\mathbf{x}, \mathbf{y})$ for $i = 1, 2, 3, 4, 5$ remain invariant under the action of GR'. Hence,

$$d(\mathbf{x}, \mathbf{y}) = d(\tau(\mathbf{x}), \tau(\mathbf{y})) = \alpha_2(\mathbf{x}, \mathbf{y}) + \alpha_3(\mathbf{x}, \mathbf{y}) + \alpha_5(\mathbf{x}, \mathbf{y}) \tag{3.10}$$

for all $\mathbf{x}, \mathbf{y} \in B(\mathbf{x}')$ and $\tau \in GR'$.

Let $\tau(B(\mathbf{x}')) = \{\tau(\mathbf{x}) : \mathbf{x} \in B(\mathbf{x}')\}$. Since each $\tau \in GR'$ is a rotation permutation around $\mathbf{1}$ that preserves distances between the elements of $B(\mathbf{x}')$, it is apparent that $\tau(B(\mathbf{x}')) = B(\tau(\mathbf{x}'))$. Let $\overline{B(\mathbf{x}')} = [2]^k$ and $\overline{\mathbf{x}}$ be obtained from $B(\mathbf{x}')$ and $\mathbf{x}$ by replacing the non-one entries with 2 respectively. Then for each $\mathbf{x}' \in \{2, \cdots, s\}$,

$$d(\mathbf{x}, \mathbf{y}) = d(\overline{\mathbf{x}}, \overline{\mathbf{y}}) \quad \text{for all } \mathbf{x}, \mathbf{y} \in B(\mathbf{x}') \tag{3.11}$$

Let GR' act on $\overline{B(\mathbf{x}')}$ by $\tau(\overline{\mathbf{x}}) = \overline{\tau(\mathbf{x})}$. Then, since $\tau(B(\mathbf{x}')) = B(\tau(\mathbf{x}'))$ by (3.11),

$$d(\tau(\mathbf{x}), \tau(\mathbf{y})) = d(\overline{\tau(\mathbf{x})}, \overline{\tau(\mathbf{y})}), \tag{3.12}$$

and

$$\{\tau(\overline{\mathbf{x}}) : \mathbf{x} \in B(\mathbf{x}')\} = \{\overline{\tau(\mathbf{x})} : \mathbf{x} \in B(\mathbf{x}')\} = \overline{\tau(B(\mathbf{x}'))} = \overline{B(\tau(\mathbf{x}'))} = [2]^k. \tag{3.13}$$

Now, (3.10), (3.11) and (3.12) lead to

$$d(\tau(\overline{\mathbf{x}}), \tau(\overline{\mathbf{y}})) = d(\overline{\tau(\mathbf{x})}, \overline{\tau(\mathbf{y})}) = d(\tau(\mathbf{x}), \tau(\mathbf{y})) = d(\mathbf{x}, \mathbf{y}) = d(\overline{\mathbf{x}}, \overline{\mathbf{y}}). \tag{3.14}$$

for all $\mathbf{x}, \mathbf{y} \in B(\mathbf{x}')$. Hence, by (3.13) and (3.14) GR' acts on $\overline{B(\mathbf{x}')} = [2]^k$ as rotation isometries. Then by Lemma 3 each element τ of GR' acting on the elements of the set $\overline{B(\mathbf{x}')}$ can be written as $\tau = \gamma^{\mathbf{x}'}$, where $\gamma^{\mathbf{x}'}$ is a permutation of columns of $\overline{\mathbf{B}(\mathbf{x}')}$. Let $\gamma^{\mathbf{x}'}$ permute the columns of $\mathbf{B}(x')$ the same way it permutes the columns of $\overline{B(\mathbf{x}')}$. Also, let

$\mu^{\mathbf{x}'} = (\gamma^{\mathbf{x}'})^{-1}\tau$ then $\mu^{\mathbf{x}'}(\overline{\mathbf{x}}) = (\gamma^{\mathbf{x}'})^{-1}\tau(\overline{\mathbf{x}}) = \overline{\mathbf{x}}$ for each $\mathbf{x} \in B(\mathbf{x}')$. This implies that $\mu^{\mathbf{x}'}$ only changes the non-one elements to non-one elements in the rows of $\mathbf{B}(\mathbf{x}')$. Since,

$$d(\mathbf{x}, \mathbf{y}) = d(\tau(\mathbf{x}), \tau(\mathbf{y})) = d((\gamma^{\mathbf{x}'})^{-1}\tau(\mathbf{x}), (\gamma^{\mathbf{x}'})^{-1}\tau(\mathbf{y})) = d(\mu^{\mathbf{x}'}(\mathbf{x}), \mu^{\mathbf{x}'}(\mathbf{y}))$$

for all $\mathbf{x}, \mathbf{y} \in B(\mathbf{x}')$, and $\mu^{\mathbf{x}'}$ fixes $\mathbf{1}$, $\mu^{\mathbf{x}'}$ is a rotation around $\mathbf{1}$ and an isometry on the set $B(\mathbf{x}')$. Let $\mu^{\mathbf{x}'}(B(\mathbf{x}')) = \cup_{i=0}^{k}\mu^{\mathbf{x}'}(R_i(\mathbf{x}'))$ and $\mu^{\mathbf{x}'}(\mathbf{B}(\mathbf{x}'))$ be the matrix whose $(\sum_{j=0}^{i-1}\binom{k}{j} + 1)$th row to $(\sum_{j=0}^{i}\binom{k}{j})$th row be the elements of $\mu^{\mathbf{x}'}(R_i(\mathbf{x}'))$ in some order, where $\mu^{\mathbf{x}'}(R_i(\mathbf{x}'))$ is the set obtained by applying $\mu^{\mathbf{x}'}$ elementwise. Now, for $j = 1, \cdots, k$, the non-one entry appearing in the j'th column of $\mathbf{B}(\mathbf{x}')$ must be mapped by $\mu^{\mathbf{x}'}$ to the same non-one element every time it appears. Otherwise, $\mu^{\mathbf{x}'}$ is not an isometry rotation around $\mathbf{1}$ on the set $B(\mathbf{x}')$. Hence, by taking $\tau^{\mathbf{x}'} = \gamma^{\mathbf{x}'}\mu^{\mathbf{x}'}$ one can deduce that there exists some $\tau^{\mathbf{x}'} \in S_{\{2,\cdots,s\}} \wr S_k$ such that

$$\tau(\mathbf{x}) = \tau^{\mathbf{x}'}(\mathbf{x})$$

for all $\mathbf{x} \in B(\mathbf{x}')$ and $\tau \in GR'$. Next, it is shown that proving the following claim is sufficient.

Claim: There exists some $\tau^{\text{full}} \in S_{\{2,\cdots,s\}} \wr S_k$ acting on $[s]^k$ such that, for each $\mathbf{x}' \in \{2, \cdots, s\}^k$, $\tau^{\text{full}}(\mathbf{x}) = \tau^{\mathbf{x}'}(\mathbf{x}) = \tau(\mathbf{x})$ for all $\mathbf{x} \in B(\mathbf{x}')$.

First, the claim implies that $\tau \in S_{\{2,\cdots,s\}} \wr S_k$ for arbitrary $\tau \in GR'$. Then, $|GR'| \leq |S_{(s-1)} \wr S_k|$. By Theorem 7, GR' contains an isomorphic copy of $S_{(s-1)} \wr S_k$ as a subgroup as $0 < t < k$ was assumed and ($k > t + 1$ or $s > 2$). Hence, $GR' = H_{k,s,t} \cong S_{(s-1)} \wr S_k$. Now, it is clear that $\Pi(\text{OAP}(k, s, t, \lambda)) = GR'_{\{\mathbf{y}:d(\mathbf{1},\mathbf{y})>t\}}$, where $GR'_{\{\mathbf{y}:d(\mathbf{1},\mathbf{y})>t\}}$ is the restriction of GR' to variables $\{\mathbf{y} : d(\mathbf{1}, \mathbf{y}) > t\}$. Finally, since GR' preserves the set $\{\mathbf{y} : d(\mathbf{1}, \mathbf{y}) > t\}$, no permutations are lost by restricting GR'. Hence, $\Pi(\text{OAP}(k, s, t, \lambda)) = GR'_{\{\mathbf{y}:d(\mathbf{1},\mathbf{y})>t\}} = H_{k,s,t} \cong S_{(s-1)} \wr S_k$.

Proof of the Claim: Let $\mathbf{x}'_1, \mathbf{x}'_2 \in \{2, \cdots, s\}^k$ be such that $\mathbf{x}'_1$ and $\mathbf{x}'_2$ are different only on the lth coordinate. Let $\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)$ be the matrix obtained by taking the elements in $B(\mathbf{x}'_1) \cap B(\mathbf{x}'_2)$ as rows in some order. Then, the lth column of $\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)$ is the column of all 1s and the jth column of $\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)$ for $j \neq l$ has only entries from the set $\{1, \mathbf{x}'_1[j]\}$, where $\mathbf{x}'_1[j] = \mathbf{x}'_2[j]$ is

the jth entry of $\mathbf{x}'_1$. Furthermore, $\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)^{-l}$ consists of all 2^{k-1} one\non-one combinations, where $\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)^{-l}$ is obtained from $\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)$ by deleting its lth column.

Now write

$$\tau(\mathbf{x}) = \begin{cases} \tau^{\mathbf{x}'_1}(\mathbf{x}) = \gamma^{\mathbf{x}'_1}\mu^{\mathbf{x}'_1}(\mathbf{x}) & \text{if } \mathbf{x} \in B(\mathbf{x}'_1), \\ \tau^{\mathbf{x}'_2}(\mathbf{x}) = \gamma^{\mathbf{x}'_2}\mu^{\mathbf{x}'_2}(\mathbf{x}) & \text{if } \mathbf{x} \in B(\mathbf{x}'_2), \end{cases} \tag{3.15}$$

where $\gamma^{\mathbf{x}'_i}(\mathbf{x})$ is a permutation of columns of $\mathbf{x}$ and $\mu^{\mathbf{x}'_i}(\mathbf{x})$ only changes the non-one elements in the columns of $\mathbf{B}(\mathbf{x}'_i)$. First, $\gamma^{\mathbf{x}'_i}, \mu^{\mathbf{x}'_i} \in S_{\{2,\cdots,s\}} \wr S_k$. Secondly,

$$\tau^{\mathbf{x}'_1}(\mathbf{x}) = \tau^{\mathbf{x}'_2}(\mathbf{x}) = \tau(\mathbf{x})$$

for all $\mathbf{x} \in \mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)$. Hence, $\mu^{\mathbf{x}'_2}\gamma^{\mathbf{x}'_2}(\mathbf{x}) = \gamma^{\mathbf{x}'_2}\mu^{\mathbf{x}'_2}(\mathbf{x}) = \mu^{\mathbf{x}'_1}\gamma^{\mathbf{x}'_1}(\mathbf{x}) = \gamma^{\mathbf{x}'_1}\mu^{\mathbf{x}'_1}(\mathbf{x})$ for all $\mathbf{x} \in \mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)$. This implies that

$$\gamma^{\mathbf{x}'_1}(\overline{\mathbf{x}}) = \gamma^{\mathbf{x}'_2}(\overline{\mathbf{x}}) \quad \text{for } \overline{\mathbf{x}} \in \overline{\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)}. \tag{3.16}$$

Since $\overline{\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)}$ is $[2]^{k-1}$ with an all 1s column attached to it, columns of $\overline{\mathbf{B}(\mathbf{x}'_1, \mathbf{x}'_2)}$ are distinguishable. Hence, equation (3.16) completely (uniquely) determines how columns of $\mathbf{x}$ are permuted by

$$\gamma^{\mathbf{x}'_1}(\mathbf{x}) = \gamma^{\mathbf{x}'_2}(\mathbf{x}) \quad \text{for } \mathbf{x} \in B(\mathbf{x}'_1) \cup B(\mathbf{x}'_2). \tag{3.17}$$

Now, one can see that

$$\mu^{\mathbf{x}'_1}(\mathbf{x}) = \mu^{\mathbf{x}'_2}(\mathbf{x}) \quad \text{for } \mathbf{x} \in B(\mathbf{x}'_1, \mathbf{x}'_2). \tag{3.18}$$

Then, $\mu^{\mathbf{x}'_1}((\mathbf{x})^{-l}) = \mu^{\mathbf{x}'_2}((\mathbf{x})^{-l})$ for each $(\mathbf{x})^{-l}$ appearing as a row of $\begin{bmatrix} \mathbf{B}(\mathbf{x}'_1)^{-l} \\ \mathbf{B}(\mathbf{x}'_2)^{-l} \end{bmatrix}$ as $B(\mathbf{x}'_1)^{-l} = B(\mathbf{x}'_2)^{-l}$, where $B(\mathbf{x}'_j)^{-l}$ is the set of rows of $\mathbf{B}(\mathbf{x}'_j)^{-l}$ for $j = 1, 2$. Now, extend the domain of $\mu^{\mathbf{x}'_1}$ from $B(\mathbf{x}'_1)$ to $B(\mathbf{x}'_1) \cup B(\mathbf{x}'_2)$ by setting

$$\mu^{\mathbf{x}'_1}_{\text{ext}}(\mathbf{x}) = \begin{cases} \mu^{\mathbf{x}'_1}(\mathbf{x}) & \text{if } \mathbf{x} \in B(\mathbf{x}'_1), \\ \mu^{\mathbf{x}'_2}(\mathbf{x}) & \text{if } \mathbf{x} \in B(\mathbf{x}'_2). \end{cases} \tag{3.19}$$

By equation (3.18) this extension is well defined.

Next, it is shown that $\mu^{\mathbf{x}_1'}_{\text{ext}}(\mathbf{x})$ is injective and that $\mu^{\mathbf{x}_1'}_{\text{ext}}(\mathbf{x}) \in S^k_{\{2,\cdots,s\}} \subset S_{\{2,\cdots,s\}} \wr S_k$. First, by the preceeding arguments, $\mu^{\mathbf{x}_1'}_{\text{ext}}$ acts on $\begin{bmatrix} \mathbf{B}(\mathbf{x}_1')^{-l} \\ \mathbf{B}(\mathbf{x}_2')^{-l} \end{bmatrix}$ as an element of $S^{k-1}_{\{2,\cdots,s\}}$. Hence, it suffices to show that $\mu^{\mathbf{x}_1'}_{\text{ext}}$ acts on $\begin{bmatrix} \mathbf{B}(\mathbf{x}_1')^{l} \\ \mathbf{B}(\mathbf{x}_2')^{l} \end{bmatrix}$ as an element of $S_{\{2,\cdots,s\}}$ and $\mu^{\mathbf{x}_1'}_{\text{ext}}$ is injective, where $\mathbf{B}(\mathbf{x}_j')^l$ is the lth column of $\mathbf{B}(\mathbf{x}_j')$. Note that $\mu^{\mathbf{x}_j'}$ fixes 1 in every column of $\mathbf{B}(\mathbf{x}_j')$. Hence, $\mu^{\mathbf{x}_1'}_{\text{ext}}$ fixes 1 in every column of $\begin{bmatrix} \mathbf{B}(\mathbf{x}_1') \\ \mathbf{B}(\mathbf{x}_2') \end{bmatrix}$ and consequently in $\begin{bmatrix} \mathbf{B}(\mathbf{x}_1')^{l} \\ \mathbf{B}(\mathbf{x}_2')^{l} \end{bmatrix}$. Furthermore, since each $\mu^{\mathbf{x}_j'}_{\text{ext}}$ must map each non-one entry in $\mathbf{B}(\mathbf{x}_j')^l$ to the same non-one element in $[s]^k$, so does $\mu^{\mathbf{x}_1'}_{\text{ext}}$. Hence, $\mu^{\mathbf{x}_1'}_{\text{ext}}$ acts on $\begin{bmatrix} \mathbf{B}(\mathbf{x}_1') \\ \mathbf{B}(\mathbf{x}_2') \end{bmatrix}$ as an element of $S^k_{\{2,\cdots,s\}}$. Now by equations (3.15), (3.17) and (3.19), one can see that

$$\mu^{\mathbf{x}_1'}_{\text{ext}}(\mathbf{x}) = (\gamma^{\mathbf{x}_1'})^{-1}\tau(\mathbf{x}) \quad \text{for } \mathbf{x} \in B(\mathbf{x}_1') \cup B(\mathbf{x}_2'). \tag{3.20}$$

Injectivity of $\mu^{\mathbf{x}_1'}_{\text{ext}}$ follows from equation (3.20). Now, extend the domain of $\tau^{\mathbf{x}_1'}(\mathbf{x})$ for $\mathbf{x} \in B(\mathbf{x}_1')$ with $B(\mathbf{x}_2')$ by defining

$$\tau^{\mathbf{x}_1'}_{\text{ext}}(\mathbf{x}) = \gamma^{\mathbf{x}_1'}\mu^{\mathbf{x}_1'}_{\text{ext}}(\mathbf{x}) \quad \text{for } \mathbf{x} \in B(\mathbf{x}_1') \cup B(\mathbf{x}_2').$$

To finish the proof, notice that there exists a finite sequence $\{\mathbf{x}_i'\}_{i=1}^m$ such that $d(\mathbf{x}_i', \mathbf{x}_{i+1}') = 1$ and

$$\bigcup_{i=1}^{m} \{\mathbf{x}_i'\} = \{2, \cdots, s\}^k,$$

where $\mathbf{x}_i'$ are not necessarily distinct. Extend the domain of $\tau^{\mathbf{x}_1'}$ along this sequence. It is understood that an extension in the domain is made only for $\mathbf{x}_i'$ appearing for the first time. Let τ_{full} be the resulting map. Since each extension is injective, acts as an element of $S_{\{2,\cdots,s\}} \wr S_k$ in its extended domain, and the domain of τ^{full} is

$$\bigcup_{i=1}^{(s-1)^k} B(\mathbf{x}_i') = [s]^k,$$

τ^{full} acts on $[s]^k$ as an element of $S_{\{2,\cdots,s\}} \wr S_k$.

□

Theorem 9. *Unless $t = 0$ or $t = k$, $G_{s,k} \cong S_s \wr S_k$ in Theorem 9 of Bulutoglu and Margot [7] is the largest subgroup of S_{s^k} that sends equations (3.3) to themselves. For $t = 0$ or $t = k$ the largest such group is S_{s^k}.*

Proof. First, consider the case when $0 < k - t < k$. Let G be the group of all coordinate permutations of $[s]^k$ which permute the rows of the constraint matrix $\mathbf{A}$ pertaining to equations (3.3) in the full space $\mathbb{R}^{s^k}$. It was shown in Bulutoglu and Margot [7] that G contains $G_{s,k}$ as a subgroup. Hence, it suffices to show that $G \leq G_{s,k}$. First, it is shown that every element in G is an isometry.

Let $\sigma \in G$ and $\mathbf{z} \in [s]^k$. Since $G_{s,k}$ acts transitively on $[s]^k$ and itself consists of isometries, one may assume $\sigma(\mathbf{z}) = \mathbf{z}$. Consider the set S of all rows of $\mathbf{A}$ in which $\mathbf{z}$ appears. Since $\sigma(\mathbf{z}) = \mathbf{z}$, σ must permute S. Let $\mathbf{w} \in [s]^k$ be such that $d(\mathbf{z}, \mathbf{w}) = i$, where $i \in \{0, 1, \cdots, k-t\}$. Then $\mathbf{w}$ appears in exactly $\binom{k-i}{t}$ rows in S. Since σ preserves S, $\sigma(w)$ also appears in exactly $\binom{k-i}{t}$ rows in S. Then one must have $d(\mathbf{z}, \sigma(\mathbf{w})) = i$ as no element is repeated in the set $\{\binom{k-i}{t}\}_{i=0}^{k-t}$. Hence, $d(\mathbf{z}, \mathbf{w}) = d(\sigma(\mathbf{z}), \sigma(\mathbf{w}))$ for all $\mathbf{z}$ and $\mathbf{w}$ such that $0 \leq d(\mathbf{z}, \mathbf{w}) \leq k - t$.

Now, for any $\mathbf{z}$ and $\mathbf{y}$ such that $d(\mathbf{z}, \mathbf{y}) = k - t + 1$ there exists $\mathbf{y}_1$ such that $d(\mathbf{z}, \mathbf{y}) = d(\mathbf{z}, \mathbf{y}_1) + d(\mathbf{y}_1, \mathbf{y})$, where $d(\mathbf{z}, \mathbf{y}_1) = k - t$ and $d(\mathbf{y}_1, \mathbf{y}) = 1 \leq k - t$. By the triangle inequality $d(\sigma(\mathbf{z}), \sigma(\mathbf{y})) \leq d(\sigma(\mathbf{z}), \sigma(\mathbf{y}_1)) + d(\sigma(\mathbf{y}_1), \sigma(\mathbf{y})) = k - t + 1 = d(\mathbf{z}, \mathbf{y})$. By repeating the same argument for $\mathbf{z}$ and $\mathbf{y}$ such that $d(\mathbf{z}, \mathbf{y}) = k - t + i$ for $i = 2, \cdots, t$,

$$d(\sigma(\mathbf{z}), \sigma(\mathbf{y})) \leq d(\mathbf{z}, \mathbf{y}) \tag{3.21}$$

for all $\mathbf{z}, \mathbf{y}$ and σ. Let $\mathbf{z}' = \sigma(\mathbf{z})$, $\mathbf{y}' = \sigma(\mathbf{y})$ and $h = \sigma^{-1}$ then

$$d(\mathbf{z}', \mathbf{y}') \leq d(h(\mathbf{z}'), h(\mathbf{y}')) \tag{3.22}$$

for all $\mathbf{z}', \mathbf{y}'$ and h. Combining inequalities (3.21) and (3.22) leads to

$$d(\sigma(\mathbf{z}), \sigma(\mathbf{y})) = d(\mathbf{z}, \mathbf{y})$$

for all $\mathbf{z}, \mathbf{y} \in [s]^k$ and $\sigma \in G$. Hence, by Lemma 3, an isomorphic copy of G is contained in $S_s \wr S_k$. Now, this implies that $G \leq G_{s,k} \cong S_s \wr S_k$. The cases $t = 0$ and $t = k$ are easy to see. □

IV. Computational Research

4.1 Chapter Introduction

As discussed in Section 2.5, Margot [23] defined the symmetry group of an LP to be the set of all permutations of the variables of an LP that send feasible points to feasible points with the same objective function value. Margot [23] and Ostrowski *et al.* [27] defined the *symmetry group* of an ILP to be the set of all permutations of the variables that map a feasible solution to a feasible solution with the same objective value. Let the symmetry group of ILP (2.4) be

$$\mathcal{G}(k,s,t) = \{\pi \in S_{s^k} |\ \pi(\mathbf{x}) \in \mathcal{F}(k,s,t)\}$$

and the symmetry group of ILP (3.2) be

$$\mathcal{G}'(k,s,t) = \{\pi \in S_{s^k-m} |\ \pi(\mathbf{x}) \in \mathcal{F}'(k,s,t)\}.$$

Let T be the set of deleted variables to get ILP (3.2) from ILP (2.4). It is not hard to see that $\text{stab}(T, G_{s,k})$ is the set of all permutations that permute indices of $\mathbf{x}'$ by permuting coordinates in $(\alpha_1^j + 1, \alpha_2^j + 1, \cdots, \alpha_k^j + 1)$ and/or by independently sending a subset of non-one coordinates to non-one coordinates. Hence $|\text{stab}(T, G_{s,k})| = (s-1)!^k k!$. Let $G(\mathbf{A}(k,s,t), \lambda\mathbf{1})^{\text{LP}}$ and $G(\mathbf{A}'(k,s,t), \mathbf{b})^{\text{LP}}$ denote the LP relaxation symmetry groups of ILP (2.4) and ILP (3.2) respectively. Let $G(\mathbf{A}, \mathbf{b})$ be as in equation (2.15) in Section 2.4 with $\mathbf{c} = \mathbf{1}$ and $\mathbf{d} = p_{\max}\mathbf{1}$. Recall from Section 3.3 that $\text{stab}(T, G_{s,k}) = G(\mathbf{A}'(k,s,t), \mathbf{b})$ and $G_{s,k} = G(A(k,s,t), \lambda\mathbf{1})$. These results imply that $|\mathcal{G}'(k,s,t)| \geq |G(\mathbf{A}'(k,s,t), \mathbf{b})| = (s-1)!^k k!$ and $|\mathcal{G}(k,s,t)| \geq \left|G_{s,k}\right| = |G(A(k,s,t), \lambda\mathbf{1})| = s!^k k!$.

In Section 3.2, it is shown that all the inequalities in the LP relaxation of ILP (3.2) are facets. Hence, by the Margot [23] symmetry group definition, this implies that $G(\mathbf{A}'(k,s,t), \mathbf{b})^{\text{LP}} = G(\mathbf{A}'(k,s,t), \mathbf{b}) \cong S_{\{2,3,\cdots,s\}} \wr S_k$. However, since ILP (2.4) has equality constraints it is not clear whether $G(\mathbf{A}(k,s,t), \lambda\mathbf{1}) = G(\mathbf{A}(k,s,t), \lambda\mathbf{1})^{\text{LP}}$ or not. The most that can be said presently is that $G(\mathbf{A}(k,s,t), \lambda\mathbf{1}) \subseteq G(\mathbf{A}(k,s,t), \lambda\mathbf{1})^{\text{LP}}$. In Section 4.2 an

efficient, practical method for computing $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ is developed and implemented. It is observed that for $1 \leq t \leq k-1$

$$\left|G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}\right| = \begin{cases} (k+1)\left|G(\mathbf{A}(k, s, t), \lambda\mathbf{1})\right| = (k+1)!2^k & s = 2 \text{ and even } t, \\ \left|G(\mathbf{A}(k, s, t), \lambda\mathbf{1})\right| = k!\,(s!)^k & \text{otherwise.} \end{cases}$$

For $s = 2$ and even t, using the larger symmetry group drastically reduces solution times. In fact, in all almost all cases considered, deleting variables from the ILP (2.5) formulation to get the ILP (3.2) formulation proved to be counterproductive. The reason for this is that for $1 \leq t \leq k-1$

$$\left|G(\mathbf{A}(k, s, t), \lambda\mathbf{1})\right| = s^k \left|G(\mathbf{A}'(k, s, t), \mathbf{b})\right|$$

and

$$\left|G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}\right| = \begin{cases} (k+1)\,2^k \left|G(\mathbf{A}'(k, s, t), \mathbf{b})\right| & s = 2 \text{ and even } t, \\ s^k \left|G(\mathbf{A}'(k, s, t), \mathbf{b})\right| & \text{otherwise.} \end{cases}$$

Hence, exploiting a larger symmetry group more than overcomes the additional computational burden of having a larger number of variables. This underscores the importance of developing tools for finding larger subgroups of the full symmetry group of an ILP.

As discussed in Section 2.5, whenever the Margot [22] isomorphism pruning algorithm is used for solving a symmetric ILP in the form of ILP (2.14) or ILP (2.16), $G(\mathbf{A}, \mathbf{b})^{\mathrm{LP}}$ or a subgroup of $G(\mathbf{A}, \mathbf{b})^{\mathrm{LP}}$ is used for isomorphism pruning. However, when the goal is to find at least one OA(N, k, s, t) for a given N, k, s, t or to prove that no OA(N, k, s, t) exists, it is viable to use $\mathcal{G}(k, s, t)$ $\left(\mathcal{G}'(k, s, t)\right)$ instead of $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ $\left(G(\mathbf{A}'(k, s, t), \mathbf{b})^{\mathrm{LP}}\right)$. This will yield significant increases in computational efficiency if $|\mathcal{G}(k, s, t)|$ $\left(|\mathcal{G}'(k, s, t)|\right)$ is much larger than $\left|G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}\right|$ $\left(\left|G(\mathbf{A}'(k, s, t), \mathbf{b})^{\mathrm{LP}}\right|\right)$. Hence, one open, fundamental question regarding ILP (2.5) is: How much larger is $|\mathcal{G}(k, s, t)|$ $\left(|\mathcal{G}'(k, s, t)|\right)$ than $\left|G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}\right|$ $\left(\left|G(\mathbf{A}'(k, s, t), \mathbf{b})^{\mathrm{LP}}\right|\right)$? A method for computing $\mathcal{G}(k, s, t)$ is given in Section 4.4. This method requires all possible solutions to ILP (2.5) as an input. Hence, it cannot be used for solving unsolved problems. However, it

is a good diagnostic test telling us how close we are to exploiting all possible symmetries of ILP (2.5) when $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ is used with Margot [22] isomorphism pruning algorithm. One interesting question that is investigated is how the number of isomorphism classes of solutions to ILP (2.5) and ILP (3.2) compare using $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ and $G(\mathbf{A}'(k, s, t), \mathbf{b})^{\mathrm{LP}}$, respectively.

In Section 4.3, the Section 4.2 method is generalized and the generalized method is tested on the MILP library problems studied in Liberti [18].

The improvements in Section 4.5 enable the use of Margot [22] isomorphism pruning solver in the algorithm in Figure 2.4 ILP formulation without losing OD classes or isomorphism classes of $\mathrm{OA}(N, k, s, t)$. Also, a previously unknown large subgroup of $G(\mathbf{A}(k, 2, t), \lambda\mathbf{1})^{\mathrm{LP}}$ is found.

All computations in this chapter were performed on an HP Z820 workstation with 64GB of RAM and a 3.10 GHz Intel Xeon E5-2687W processor.

4.2 Research Objective 1

4.2.1 Computing $G(\boldsymbol{A}(k, 2, t), \lambda\mathbf{1})^{\mathrm{LP}}$.

Every solution to the LP relaxation of ILP (2.4) can be written in the form

$$p_{\max}\mathbf{1} \geq \mathbf{x} = \frac{\lambda}{s^{k-t}}\mathbf{1} + \mathbf{v} \geq 0$$

for some $\mathbf{v} \in \mathrm{Null}(\mathbf{A}(k, s, t)) \bigcap [\frac{-\lambda}{s^{k-t}}, p_{\max} - \frac{\lambda}{s^{k-t}}]^{s^k}$. This is true because $s^{t-k}\mathbf{1}$ is a particular solution and $\mathbf{x} \geq \mathbf{0}$. Since any permutation of s^k coordinates preserves $[\frac{-\lambda}{s^{k-t}}, p_{\max} - \frac{\lambda}{s^{k-t}}]^{s^k}$, $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ is the set of all permutations in S_{s^k} that preserve the elements of $\mathrm{Null}(\mathbf{A}(k, s, t))$. $\mathrm{Null}(\mathbf{A}(k, s, t))$ is the orthogonal complement of the row space of $\mathbf{A}(k, s, t)$, hence the same set is also characterized as the set of all permutations that stabilize the row space of $\mathbf{A}(k, s, t)$. Such a set is computed as the automorphism group of $\mathbf{P}_{\mathbf{A}^T(k,s,t)}$, where

$$\mathbf{P}_{\mathbf{A}^T(k,s,t)} = \mathbf{A}^T(k, s, t)\left(\mathbf{A}(k, s, t)\mathbf{A}^T(k, s, t)\right)^+ \mathbf{A}(k, s, t) \tag{4.1}$$

is the orthogonal projection matrix on to the row space of $\mathbf{A}(k, s, t)$, and $\left(\mathbf{A}(k, s, t)\mathbf{A}^T(k, s, t)\right)^+$ is the Moore-Penrose pseudoinverse of $\mathbf{A}(k, s, t)\mathbf{A}^T(k, s, t)$ [17]. The automorphism group of $\mathbf{P}_{\mathbf{A}^T(k,s,t)}$ is the set of all $\pi \in S_{s^k}$ that send $\mathbf{P}_{\mathbf{A}^T(k,s,t)}$ to itself when rows and columns of $\mathbf{A}^T(k, s, t)$ are permuted according to π. This automorphism group is computed as the automorphism group of an edge colored graph with s^k vertices. Each distinct entry in $\mathbf{P}_{\mathbf{A}^T(k,s,t)}$ is labeled with a distinct color. There is an edge between ith and jth vertices labeled with color l, if and only if the (i, j)th entry of $\mathbf{P}_{\mathbf{A}^T(k,s,t)}$ is labeled with color l.

Let $\mathbf{A}(k, s, t)$ be $m \times n$ (where $n = s^k$), $p = \text{rank}\,(\mathbf{A}(k, s, t))$, and $\mathbf{A}(k, s, t) = \mathbf{U}\mathbf{D}\mathbf{V}^T$ be the singular value decomposition of $\mathbf{A}(k, s, t)$. Then, by using the results in [17], equation (4.1) simplifies to

$$\mathbf{P}_{\mathbf{A}^T(k,s,t)} = \mathbf{V}\mathbf{I}_n^{(p)}\mathbf{V}^T, \tag{4.2}$$

where

$$\mathbf{I}_n^{(p)} = \begin{bmatrix} \mathbf{I}_{p\times p} & \mathbf{0}_{p\times(n-p)} \\ \mathbf{0}_{(n-p)\times p} & \mathbf{0}_{(n-p)\times(n-p)} \end{bmatrix}.$$

Equation (4.2) was used to calculate $\mathbf{P}_{\mathbf{A}^T(k,s,t)}$ as it requires fewer floating point operations, leading to improved accuracy.

The automorphism group of $\mathbf{P}_{\mathbf{A}^T(k,s,t)}$ was computed by using Nauty, where edge coloring was implemented as described in McKay [25]. $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\text{LP}}$ was computed for many k, s, t combinations by using the aforementioned method. In all cases considered, $G(\mathbf{A}(k, s, t), \lambda\mathbf{1}) \subsetneq G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\text{LP}}$ for $s = 2$ and even t, and $G(\mathbf{A}(k, s, t), \lambda\mathbf{1}) = G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\text{LP}}$ otherwise. Consequently, for $s = 2$ and even t, using the group $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\text{LP}}$ significantly decreased solution times. This method generalizes the concept of $G(\mathbf{A}, \mathbf{b})^{\text{LP}}$ first defined in Margot [23]. A further generalization of this method that finds $G(\mathbf{A}, \mathbf{b})^{\text{LP}}$ for arbitrary $\mathbf{A}$ and $\mathbf{b}$, where a particular solution with equal coordinates is not available, is developed in Section 4.3.

ILP (2.5) was solved for many k, s, t combinations using the groups $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})$ and $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\text{LP}}$. This is legitimate as ILPs (2.4) and (2.5) have the same feasible

set. ILP (3.2) was solved for the same k, s, t combinations using $G(\mathbf{A}'(k, s, t), \mathbf{b})^{\mathrm{LP}}$. A speed comparison of these three formulations using Margot ILP solver [22] is made in Table 4.1. For each OA(N, k, s, t), the second, third and fourth columns report the number of solutions enumerated for ILP (2.5) using $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})$, ILP (2.5) using $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}})$, and ILP (3.2) using $G(\mathbf{A}'(k, s, t), \mathbf{b})^{\mathrm{LP}}$. Likewise, the fifth, sixth and seventh columns report the time it took to enumerate these solutions. Even though ILP (3.2) has fewer variables, computational experiments summarized in Table 4.1 suggest that it should not be preferred over ILP (2.5). It is evident from Table 4.1 that exploiting the larger symmetry group more than overcomes the additional computational burden of having a larger number of variables. In fact, the computational savings appear to grow exponentially with the number of variables. On the other hand, the cases OA$(64, 7, 2, 4)$ and OA$(24, 11, 2, 3)$ do buck this trend.

Table 4.1: Formulation Comparisons

OA(N, k, s, t)	ILP (2.5) $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})$ # Designs	ILP (2.5) $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ # Designs	ILP (3.2) $G(\mathbf{A}'(k, s, t), \mathbf{b})^{\mathrm{LP}}$ # Designs	ILP (2.5) $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})$ Times (sec.)	ILP (2.5) $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ Times (sec.)	ILP (3.2) $G(\mathbf{A}'(k, s, t), \mathbf{b})^{\mathrm{LP}}$ Times (sec.)
OA(20,6,2,2)	75	23	3069	1.42	6.74	63.99
OA(20,7,2,2)	474	102	51695	13.4	9.22	2578.82
OA(20,8,2,2)	1603	211	383729	108.96	21.98	66377
OA(20,9,2,2)	2477	351	1157955	484.55	66.91	879382
OA(20,10,2,2)	2389	260	$\geq$ 28195	1683.95	215.24	$\geq$ 37214
OA(24,5,2,2)	63	31	723	1.07	10.06	18.36
OA(24,6,2,2)	1350	274	62043	22.03	12.05	1381.39
OA(24,7,2,2)	57389	7990	6894001	1720.96	257.27	428220
OA(24,8,2,2)	1470157	165596	4505018	99738	10082	653671
OA(24,9,2,2)	3815882	1309475	-	763643	223138	-
OA(24,5,2,3)	1	1	2	0.13	6.38	11.64
OA(24,6,2,3)	2	2	5	0.25	6.62	11.67
OA(24,7,2,3)	1	1	5	0.32	9.09	16.04
OA(24,8,2,3)	1	1	6	1	14.11	22.88
OA(24,9,2,3)	1	1	6	5.9	25.9	44.02
OA(24,10,2,3)	1	1	5	55.49	103.59	128.95

Continued on next page

Table 4.1 – continued from previous page

OA(N, k, s, t)	ILP (2.5) $G(\mathbf{A}(k,s,t), \lambda\mathbf{1})$ # Designs	ILP (2.5) $G(\mathbf{A}(k,s,t), \lambda\mathbf{1})^{\text{LP}}$ # Designs	ILP (3.2) $G(\mathbf{A}'(k,s,t), \mathbf{b})^{\text{LP}}$ # Designs	ILP (2.5) $G(\mathbf{A}(k,s,t), \lambda\mathbf{1})$ Times (sec.)	ILP (2.5) $G(\mathbf{A}(k,s,t), \lambda\mathbf{1})^{\text{LP}}$ Times (sec.)	ILP (3.2) $G(\mathbf{A}'(k,s,t), \mathbf{b})^{\text{LP}}$ Times (sec.)
OA(24,11,2,3)	1	1	3	519.62	540	460.59
OA(32,6,2,3)	10	10	31	1.85	7.89	12.2
OA(32,7,2,3)	17	17	76	1.82	8.21	16.13
OA(32,8,2,3)	33	33	194	6.59	13.97	77.49
OA(32,9,2,3)	34	34	364	23.75	33.24	658.38
OA(32,10,2,3)	32	32	561	102.39	112.39	7338
OA(32,11,2,3)	22	22	≥ 441	560.29	597	≥ 36463
OA(40,6,2,3)	9	9	65	0.52	6.66	12.92
OA(40,7,2,3)	25	25	580	2.01	8.5	40.68
OA(40,8,2,3)	105	105	6943	19.71	27.16	4178
OA(40,9,2,3)	213	213	43713	206.25	215.22	260919
OA(40,10,2,3)	353	353	≥ 1511	1764.73	1693.85	≥ 36279
OA(48,6,2,3)	45	45	355	2.01	8.12	18.27
OA(48,7,2,3)	397	397	13469	33.73	40.11	862.1
OA(48,8,2,3)	8383	8383	896963	2231.77	2237.34	552154
OA(54,5,3,3)	4	4	49	1.9	10.26	36.01
OA(54,6,3,3)	0	0	0	17.14	36.84	167.07
OA(56,6,2,3)	86	86	1393	4.44	10.88	36.02
OA(56,7,2,3)	4049	4049	285184	443.4	449.78	20415
OA(64,7,2,4)	7	4	21	98.83	259.84	15.45
OA(64,8,2,4)	3	2	10	12.17	37.58	23.39
OA(80,6,2,4)	1	1	6	0.52	6.82	11.86
OA(80,7,2,4)	0	0	0	0.37	7.97	15.01
OA(81,5,3,4)	1	1	2	15.75	22.8	19.56
OA(96,7,2,4)	4	2	31	3.14	9.75	15.41
OA(96,8,2,4)	0	0	0	2.28	10.73	60.39
OA(112,6,2,4)	3	2	25	1.24	7.57	12.7
OA(112,7,2,4)	0	0	0	1.24	7.74	17.36
OA(144,8,2,4)	20	7	3392	1792.82	774.49	1535314
OA(162,6,3,4)	0	0	0	19.8	31.93	266.8

The code used to compute column 2 of Table 4.1 is found in Bulutoglu and Ryan [8]. The code used to compute column 3 via modification of the Bulutoglu and Ryan [8] code is

found in Section B.1. The code used to compute column 4 via modification of the Bulutoglu and Ryan [8] is found in Section B.2.

4.2.2 Finding a Large Subgroup of $G(\boldsymbol{A}(k,2,t),\lambda\mathbf{1})^{\mathrm{LP}}$.

Switch to ± 1 coding of the 2^k full factorial design using the function

$$(x_1, x_2, \cdots, x_k)^T \xrightarrow{\phi_1} ((-1)^{x_1}, (-1)^{x_2}, \cdots, (-1)^{x_k})^T$$

if $(x_1, x_2, \cdots, x_k)^T \in \{0,1\}^k$ and

$$(x_1, x_2, \cdots, x_k)^T \xrightarrow{\phi_2} \left((-1)^{x_1-1}, (-1)^{x_2-1}, \cdots, (-1)^{x_k-1}\right)^T$$

if $(x_1, x_2, \cdots, x_k)^T \in \{1,2\}^k$. Both ϕ_1 and ϕ_2 are invertible functions. So, we can switch between $\{0,1\}$, $\{1,2\}$ and $\{\pm 1\}$ codings as necessary. Let $N^x_{\mathbf{x}}$, $N^{x'}_{\mathbf{x}'}$ and $N^{x''}_{\mathbf{x}''}$ be the number of times $\mathbf{x} \in \{0,1\}^k$, $\mathbf{x}' \in \{1,2\}^k$ and $\mathbf{x}'' \in \{\pm 1\}^k$ appears in a sought after k-factor factorial design. Let $\mathbf{N}^x$, $\mathbf{N}^{x'}$ and $\mathbf{N}^{x''}$ be the $s^k \times 1$ vectors whose $\mathbf{x}$th, $\mathbf{x}'$th and $\mathbf{x}''$th variables are $N^x_{\mathbf{x}}$, $N^{x'}_{\mathbf{x}'}$ and $N^{x''}_{\mathbf{x}''}$. Functions ϕ_1 and ϕ_2 are chosen so as to preserve the ordering of the variables in $\mathbf{N}^x$, $\mathbf{N}^{x'}$ and $\mathbf{N}^{x''}$ in such a way that 0, 1 and 1 are the low levels in the $\{0,1\}$, $\{1,2\}$ and $\{\pm 1\}$ codings.

Let $\mathbf{X}''^k_{2^k} = \left(x''_{ij}\right)$ be the full factorial 2^k design in $\{-1,1\}$ coding. For an indicator vector $\mathbf{N}^{x''} \in \mathbb{R}^{2^k}$ with $\mathbf{N}^{x''} \geq \mathbf{0}$, define

$$\begin{aligned} J_{\{j_1,j_2,\cdots,j_p\}} &= \sum_{i=1}^{2^k} x''_{ij_1} x''_{ij_2} \cdots x''_{ij_p} N^{x''}_{(x''_{i1},x''_{i2},\cdots,x''_{ik})}, \\ J_{\emptyset} &= \sum_{i=1}^{2^k} N^{x''}_{(x''_{i1},x''_{i2},\cdots,x''_{ik})}. \end{aligned} \tag{4.3}$$

Then, $J_{\{j_1,j_2,\cdots,j_p\}}$ is called the *J-characteristic* of length p of $\mathbf{N}^x$ [32].

Theorem 10. *$\mathbf{N}^{x''}$ is the indicator vector of an OA$(N,k,2,t)$ if and only if $\mathbf{N}^{x''} \in \mathbb{Z}^{2^k}_{\geq 0}$ and*

$$\begin{aligned} J_{\emptyset} &= \sum_{i=1}^{2^k} N^{x''}_{(x''_{i1},x''_{i2},\cdots,x''_{ik})} = N, \\ J_{\{j_1,j_2,\cdots,j_p\}} &= \sum_{i=1}^{2^k} x''_{ij_1} x''_{ij_2} \cdots x''_{ij_p} N^{x''}_{(x''_{i1},x''_{i2},\cdots,x''_{ik})} = 0, \end{aligned} \tag{4.4}$$

for all size p subsets $\{j_1, j_2, \cdots, j_p\}$ *of* $\{1, 2, \cdots, k\}$ *and all* $p \in \{1, 2, \cdots, t\}$ *[32].*

For each equation in (4.4) whose right hand side is zero, multiply both sides by -1 and append it to (4.4). Let $G^J_{k,t}$ be the automorphism group of the resulting system of equations. Theorem 10 implies that every equation in ILPs (2.4) and (2.5) is a linear combination of equations in (4.4). Hence, $G^J_{k,t} \leq G(\mathbf{A}(k, 2, t), \lambda\mathbf{1})^{\mathrm{LP}}$. $S_2 \wr S_k$ acts on $\mathbf{x}''$ as signed permutations of columns of $\mathbf{x}''$, where $\mathbf{x}''$ is a row of $\mathbf{X}''^k_{2^k}$. For a given indicator vector $\mathbf{N}^{x''}$, this action extends to an action on J-characteristics of $\mathbf{N}^{x''}$ via equation (4.3). $S_2 \wr S_k$ acts as signed permutations on J-characteristics of $\mathbf{N}^{x''}$ as well. Hence, $S_2 \wr S_k \leq G^J_{k,t}$.

For each row $\mathbf{x}'' = (x''_1, x''_2, \cdots, x''_k)$ of $\mathbf{X}''^k_{2^k}$, let

$$a_f\mathbf{x}'' = (x''_f x''_1, x''_f x''_2, \cdots, x''_f x''_{f-1}, x''_f, x''_f x''_{f+1}, \cdots, x''_f x''_k),$$

for $1 \leq f \leq k$. Then, a_f permutes the variables $N_{\mathbf{x}''}$ by

$$N_{\mathbf{x}''} \xrightarrow{a_f} N_{a_f\mathbf{x}''}.$$

This action extends to an action on J-characteristics of $\mathbf{N}^{x''}$, where

$$J_{\{j_1,j_2,\cdots,j_p\}} \xrightarrow{a_f} \sum_{i=1}^{2^k} \left(x''_{if}\right)^p x''_{ij_1} x''_{ij_2} \cdots x''_{ij_p} N_{(x''_{if}x''_{i1}, x''_{if}x''_{i2}, \cdots, x''_{if}x''_{i(f-1)}, x''_{if}, x''_{if}x''_{i(f+1)}, \cdots, x''_{if}x''_{ik})}.$$

Each a_f acts as permutations of J-characteristics of $\mathbf{N}^{x''}$. Now, the following theorem follows.

Theorem 11. *The action of* a_f *sends a J-characteristic of length p to a J-characteristic of length p if p is even and to a J-characteristic of length* $p + 1$ *if p is odd.*

The following corrolary immediately follows from Theorem 11.

Corrolary 1. *The action of* a_f *sends the extended system of equations (4.4) to itself if t is even. Hence, if t is even,* $a_f \in G^J_{k,t}$ *and consequently* $\langle a_1, a_2, \cdots, a_k, S_2 \wr S_k\rangle \leq G^J_{k,t}$*, where* $\langle a_1, a_2, \cdots, a_k, S_2 \wr S_k\rangle$ *is the group generated by* $a_1, a_2, \cdots, a_k$ *and* $S_2 \wr S_k$*.*

Theorem 12. $|\langle a_1, a_2, \cdots, a_k, S_2 \wr S_k \rangle| = (k+1)!2^k$

Proof. Let $G_k = \langle a_1, a_2, \cdots, a_k, S_2 \wr S_k \rangle$. Let $\mathbf{D} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k]$ be a k factor, 2 level, N run generic design. Let $\boldsymbol{O}$ be the orbit of of $\mathbf{D}$ under the action of G_k. Each element of G_k is completely determined by its action on $\mathbf{D}$. Hence $|G_k| = |\boldsymbol{O}|$. For two vectors, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$, let $\mathbf{x} \odot \mathbf{y} = (x_1 y_1, x_2 y_2, \cdots, x_k y_k)^T$. Let

$$B_0 = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\}$$

and for $i = 1, 2, \cdots, k$ let

$$B_i = \{\mathbf{x}_1 \odot \mathbf{x}_i, \mathbf{x}_2 \odot \mathbf{x}_i, \cdots, \mathbf{x}_{i-1} \odot \mathbf{x}_i, \mathbf{x}_i, \mathbf{x}_{i+1} \odot \mathbf{x}_i, \cdots, \mathbf{x}_i \odot \mathbf{x}_k\}.$$

Then, for each $i = 0, 1, 2, \cdots, k$ there exists an element in $\boldsymbol{O}$ whose set of columns is B_i. Then each $2^k k!$ distinct signed permutations of the columns in B_i is a distinct element in $\boldsymbol{O}$. Hence, $|\boldsymbol{O}| \geq 2^k (k+1)!$. Observing that the set of columns (up to multiplication by ± 1) of each element in $\boldsymbol{O}$ is equal to B_i for some $i \in \{0, 1, \cdots, k\}$ finishes the proof. □

Hence, for even t and $s = 2$, $(k+1)!2^k \leq \left|G^J_{k,t}\right|$. Based on computer observations, the following conjecture is made:

Conjecture 1.

$$G^J_{k,t} = \begin{cases} \langle a_1, a_2, \cdots, a_k, S_2 \wr S_k \rangle & \textit{if } t \textit{ is even and } 1 \leq t \leq k, \\ S_2 \wr S_k & \textit{if } t \textit{ is odd and } 1 \leq t \leq k, \\ S_{2^k} & \textit{otherwise.} \end{cases}$$

Hence,

$$|G^J_{k,t}| = \begin{cases} (k+1)!2^k & \textit{if } t \textit{ is even and } 1 \leq t \leq k, \\ k!2^k & \textit{if } t \textit{ is odd and } 1 \leq t \leq k, \\ 2^k! & \textit{otherwise.} \end{cases}$$

4.3 Research Objective 2

Let $\mathbf{A}$ be the $m \times n$ constraint matrix of an ILP in the form of ILP (2.14) and $\mathbf{b}$ be the right hand side. Let $G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ be the automorphism group of $\mathbf{P}_{\mathbf{A}^T} = \mathbf{A}^T \left(\mathbf{A}\mathbf{A}^T\right)^+ \mathbf{A}$ that preserves $\mathbf{c}$ and $\mathbf{d}$. Let $\mathcal{B} = \{e_1, e_2, \cdots, e_n\}$ be the standard orthonormal basis of $\mathbb{R}^n$ and $O_1, O_2, \cdots, O_r$ be the orbits of $G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ in $\mathcal{B}$. The fixed subspace of $\mathbb{R}^n$ under the action of $G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ is defined as

$$\text{Fix}_{G(\mathbf{A},\mathbf{0},\mathbf{c},\mathbf{d})^{\text{LP}}}(\mathbb{R}^n) := \{\mathbf{x} \in \mathbb{R}^n | \gamma\mathbf{x} = \mathbf{x} \text{ for all } \gamma \in G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}\}.$$

Lemma 3 in Bödi *et al.* [4] implies that

$$\text{Fix}_{G(\mathbf{A},\mathbf{0},\mathbf{c},\mathbf{d})^{\text{LP}}}(\mathbb{R}^n) = \text{Span}\left(\beta(O_1), \beta(O_2), \cdots, \beta(O_r)\right),$$

where $\beta(S) = (\sum_{\mathbf{v} \in S} \mathbf{v})/|S|$. Let the rows of $\mathbf{E}$ be a basis for $\text{Span}\left(\beta(O_1), \beta(O_2), \cdots, \beta(O_r)\right)^{\perp}$. If the LP relaxation set intersected with the fixed space,

$$T^{\text{LP}}_{\text{Fix}_{G(\mathbf{A},\mathbf{0},\mathbf{c},\mathbf{d})^{\text{LP}}}} = \{\mathbf{x} \in \mathbb{Z}^n_{\geq 0} | \mathbf{E}\mathbf{x} = \mathbf{0} \text{ and } \mathbf{x} \text{ satisfies equations (2.14)}\},$$

is non-empty then $G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}} = G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}}$. This follows from the argument in the beginning of Section 4.2.

On the other hand, if $T^{\text{LP}}_{\text{Fix}_{G(\mathbf{A},\mathbf{0},\mathbf{c},\mathbf{d})^{\text{LP}}(\mathbb{R}^n)}}$ is empty, one must have $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \subseteq G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}} \subsetneq G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}$, where $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ is defined as in Section 2.4. To find $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ in this case, let

$$G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}} = \bigcup_{i=1}^{p} G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d}) g_i G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$$

be the double coset decomposition of $G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ using the subgroup $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$. Then, as discussed in Bremner *et al.* [5], either $(G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d}) g_i G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})) \bigcap G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}} = \emptyset$ or $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d}) g_i G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \subset G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}}$. Let $G_{\text{ext}} = \langle g_1, G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \rangle$ be the group generated by g_1 and $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$. Calculate $\text{Fix}_{G_{\text{ext}}}(\mathbb{R}^n)$ and $T^{\text{LP}}_{\text{Fix}_{G_{\text{ext}}}}$, as described in the previous paragraph, by replacing $G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ with G_{ext}. If $T^{\text{LP}}_{\text{Fix}_{G_{\text{ext}}}}$ is non-empty update

$G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ with G_{ext}. Repeat the same procedure with $G_{\text{ext}} = \langle g_i,\ G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})\rangle$ for $i = 2, \cdots, p$. The resulting $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ in the end is equal to $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}}$.

This method was first tested on the OA(N, k, s, t) cases listed in Table 4.1 by using both the constraints (2.2) with a $\mathbf{0}$ objective function as well as using ILP (2.4). As expected, this method found $G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}} = G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ in all cases. To further test this method, a number of library MILPs featured in Liberti [18] were studied. Each inequality constraint was converted to an equality constraint by adding a slack variable. Each added slack variable was labeled to be to be an integer variable if the integrality of the variable could be deduced from the constraint to which it was added. The results of that study are listed in Table 4.2. The $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ groups sizes found are the same as those found by Liberti [18]. The code used to generate this table is found in Section B.6.

Table 4.2: $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ of Liberti [18] Problems Via Double Coset Decomposition

Problem	$\left\lvert G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}\right\rvert$	$\left\lvert G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}}\right\rvert$	$\lvert G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})\rvert$	$G(\mathbf{A}, \mathbf{0}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ Times (sec.)	$G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\text{LP}}$ Times (sec.)	$G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ Times (sec.)
air03	268435456	268435456	8192	189000	-	1710
arki001	6.36×10^{61}	6.36×10^{61}	5.23×10^{44}	1270	-	433
blend2	362880	362880	362880	99.2	-	19.7
enigma	2	2	240	4.2	-	0.705
gen	2	2	2	921	-	125
mas74	4	4	4	10.8	-	1.81
mas76	4	4	4	10.2	-	1.88
misc03	48	48	12	19	-	3.69
misc06	3456000	1728000	1728000	17200	34.9	211
misc07	48	48	6	77.3	-	12.1
mzzv11	1.26×10^{56}	1.26×10^{56}	4.57×10^{44}	3390000	-	22900
mzzv42z	1.83×10^{53}	1.83×10^{53}	1.3×10^{32}	3040000	-	25700
noswot	2	2	2	38.7	-	6.25
opt1217	2	2	2	570	-	16.8

Continued on next page

Table 4.2 – continued from previous page

Problem	$\left\|G(\mathbf{A},\mathbf{0},\mathbf{c},\mathbf{d})^{\mathrm{LP}}\right\|$	$\left\|G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})^{\mathrm{LP}}\right\|$	$\|G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})\|$	$G(\mathbf{A},\mathbf{0},\mathbf{c},\mathbf{d})^{\mathrm{LP}}$ Times (sec.)	$G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})^{\mathrm{LP}}$ Times (sec.)	$G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})$ Times (sec.)
p0201	4	4	4	42.4	-	5.98
p2756	5.15×10^{10}	536870912	536870912	5320	3360	403
protfold	4	4	4	55600	-	584
qiu	24	24	24	2600	-	194
rgn	120	120	120	12.1	-	1.43
rout	120	120	120	288	-	33.4
seymour	2.78×10^{232}	2.78×10^{232}	2.78×10^{232}	811000	-	1540
stein27	303264	303264	303264	7.36	-	2.01
swath	4.24×10^{1021}	4.24×10^{1021}	3.36×10^{816}	87700	-	3850
timtab1	8	2	2	67.3	26.6	7.59
timtab2	256	2	2	265	628	20.8

In this set of problems, $G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})^{\mathrm{LP}}$ is either $G(\mathbf{A},\mathbf{0},\mathbf{c},\mathbf{d})^{\mathrm{LP}}$ or $G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})$, with $G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})^{\mathrm{LP}} = G(\mathbf{A},\mathbf{0},\mathbf{c},\mathbf{d})^{\mathrm{LP}}$ in most of the cases. In many of these cases, $\left|G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})^{\mathrm{LP}}\right| > |G(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{d})|$. Hence, this theory reveals hidden symmetries that could not otherwise be detected. If a mixed integer solver with isomorphism pruning is used on these problems, it is expected that exploiting the larger groups would overcome the computational burden of added slack variables to convert all constraints to equalities.

This method can also be applied to solve ILP (2.10). However, this requires adding $n^2 - 1$ binary slack variables. This increases the total number of binary variables in ILP (2.10) to $(3n + 11)(n - 1)/2$. Holzmann *et al.* [15] found all nonequivalent Williamson matrices up to order $n = 59$. The next open case is $n = 61$, which would require solving an ILP with 5,820 variables. The computer used for this research was unable to find $G(\mathbf{A}(12, 2, 2), \mathbf{0})^{\mathrm{LP}}$ for ILP (2.4), which is an ILP with 4,096 variables. Therefore, the first open case, $n = 61$, is beyond the reach of the current available resources. The

nonlinear form of this ILP has only 240 variables and it may be possible to solve that version. However, binary nonlinear programming is beyond the scope of this dissertation.

4.4 Research Objective 3

For fixed k, s and t, the number of non-isomorphic OA$(\lambda s^t, k, s, t)$ grows exponentially with λ; see Bulutoglu and Margot [7] and Bulutoglu and Ryan [8]. This makes it impossible to enumerate all non-isomorphic OA$(\lambda s^t, k, s, t)$ for k close to $k_{\max}(N, s, t)$ and large λ using any of the extension algorithms of Bulutoglu and Ryan [8], since these algorithms require finding all non-isomorphic OA$(\lambda s^t, i, s, t)$ for $i \leq k$. On the other hand, all non-isomorphic OA$(\lambda s^t, k, s, t)$ can be enumerated by solving ILP (2.4) with the Margot [22] Branch-and-Cut algorithm with isomorphism pruning using group $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$. However, this approach also fails for $k \geq 12$ or large λ, as it suffers from the exponential growth of the Branch-and-Cut enumeration tree with k and λ.

If $|\mathcal{G}(k, s, t)|/|G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}|$ is large, using $\mathcal{G}(k, s, t)$ instead of $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ will significantly reduce the Branch-and-Cut enumeration tree without compromising the correctness of the answer to the feasibility question of an OA$(\lambda s^t, k, s, t)$. For $i = t, t+1, \cdots, k-1$, a permutation $\pi^i \in \mathcal{G}(i, s, t)$ extends to a permutation $\pi^k \in \mathcal{G}(k, s, t)$ if and only if the action of permutation π^k on the first i factors of each of s^k possible k factor level combinations is identical to that of π^i. Using $\mathcal{G}(i, s, t)$ instead of $G(\mathbf{A}(i, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ for $i = t, t+1, \cdots, k$ (when $|\mathcal{G}(i, s, t)|/|G(\mathbf{A}(i, s, t), \lambda\mathbf{1})^{\mathrm{LP}}|$ is large) will also improve the efficiency of extension algorithm in Figure 2.4 significantly by decreasing the enumeration tree sizes, the number of solutions to the ILPs to be solved and consequently the total number of ILPs to be solved. However, using $\mathcal{G}(i, s, t)$ may cause the algorithm in Figure 2.4 to incorrectly declare an OA$(\lambda s^t, k, s, t)$ to be infeasible. The correctness of the answer to the feasibility question of an OA$(\lambda s^t, k, s, t)$ will not be compromised if and only if each permutation in $\mathcal{G}(i, s, t)$ extends to a permutation in $\mathcal{G}(k, s, t)$.

We computed $\mathcal{G}(i,s,t)$ for $i \leq k$ for many k,s,t combinations to determine $|\mathcal{G}(i,s,t)|/|G(\mathbf{A}(i,s,t),\lambda\mathbf{1})^{\mathrm{LP}}|$. This was accomplished as follows:

1. Find the indicator vectors of all OA$(\lambda s^t,i,s,t)$ either by finding all solutions to ILP (2.4) or by generating them from a set of all non-isomorphic OA$(\lambda s^t,i,s,t)$.

2. Let $\mathbf{M}$ be a $n\times s^i$ matrix whose rows are the indicator vectors of all OA$(\lambda s^t,i,s,t)$, where n is the number of all OA$(\lambda s^t,i,s,t)$.

3. Compute $G(\mathbf{M},\mathbf{1},\mathbf{1},\mathbf{1}) = \mathcal{G}(i,s,t)$ by computing the automorphism group of a vertex colored bipartite graph (the same way $G(\mathbf{A},\mathbf{1},\mathbf{1},\mathbf{1})$ in equation (2.15) is computed by Margot [23]). If $\mathbf{M}$ is not a binary matrix, then an edge colored vertex colored bipartite graph, where each color represents a distinct value in $\mathbf{M}$, must be used. Edge coloring of such a graph was implemented as described in the Nauty software documentation [25].

For many non-trivial OA$(\lambda s^t,i,s,t)$ cases, $\mathbf{M}$ is huge. Hence, for such cases Step 3 is disk space intensive. The following theorem provides a more efficient way of determining $\mathcal{G}(i,s,t)$ by converting this problem to finding the automorphism group of $\mathbf{B}^T\left(\mathbf{B}\mathbf{B}^T\right)^+\mathbf{B}$, where $\mathbf{B}$ is a basis of the row space of $\left(\mathbf{M}-\frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right)$, and $\mathbf{J}_n^{s^i}$ is a $n\times s^i$ matrix of 1's:

Theorem 13. *Let $\boldsymbol{M}$ be the $n\times s^i$ matrix above. Let the rows of $\mathbf{B}$ be a basis for the row space of $\boldsymbol{M}-\frac{\lambda}{s^{k-t}}\boldsymbol{J}_n^{s^i}$. Then the automorphism group of $\boldsymbol{P}_{\mathbf{B}^T}=\mathbf{B}^T\left(\mathbf{B}\mathbf{B}^T\right)^+\mathbf{B}$, $G_{\boldsymbol{P}_{\mathbf{B}^T}}$ is equal to $\mathcal{G}(i,s,t)$*

Proof. By the argument in Section 4.2, the automorphism group of $\mathbf{P}_{\mathbf{B}^T}$ is the set of all permutations in S_{s^i} that preserve $\mathcal{R}\left(\left(\mathbf{M}-\frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right)^T\right)$, the row space of $\mathbf{M}-\frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}$. Clearly, $\mathcal{R}\left(\left(\mathbf{M}-\frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right)^T\right)$ is a subspace of Null$(\mathbf{A}(k,s,t))$. Every solution to ILP (2.4) has the form $\mathbf{x}=\mathbf{v}+\frac{\lambda}{s^{k-t}}\mathbf{1}_{s^i}$ for some $\mathbf{v}\in\mathcal{R}\left(\left(\mathbf{M}-\frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right)^T\right)\subseteq$ Null$(\mathbf{A}(k,s,t))$. For $g\in\mathcal{G}(i,s,t)$, $\mathbf{x}=\mathbf{v}+\frac{\lambda}{s^{k-t}}\mathbf{1}_{s^i}$ and $g\mathbf{x}=g\mathbf{v}+\frac{\lambda}{s^{k-t}}\mathbf{1}_{s^i}$ both solve ILP (2.4). Hence, $g\mathbf{v}\in\mathcal{R}\left(\left(\mathbf{M}-\frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right)^T\right)$ and

$\mathcal{G}(i,s,t) \subseteq G_{\mathbf{P}_{\mathbf{B}^T}}$. Furthermore, $g\mathbf{x} = g\mathbf{v} + \frac{\lambda}{s^{k-t}}\mathbf{1}_{s^i}$ also solves ILP (2.4) for $g \in G_{\mathbf{P}_{\mathbf{B}^T}}$. Hence, $G_{\mathbf{P}_{\mathbf{B}^T}} \subseteq \mathcal{G}(i,s,t)$. □

The entries in Table 4.3 were computed as follows:

1. Compute a basis $\mathbf{B}$ for the row space of $\left(\mathbf{M} - \frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right)$ via successive applications of Q-R decomposition, taking 100 vectors at a time. Report $\operatorname{rank}\left(\mathbf{M} - \frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right) = \operatorname{rank}(\mathbf{B})$.
2. Calculate $\mathbf{P}_{\mathbf{B}^T} = \mathbf{B}^T\left(\mathbf{B}\mathbf{B}^T\right)^+\mathbf{B}$.
3. Convert $\mathbf{P}_{\mathbf{B}^T}$ to an edge colored graph with s^i vertices as described in Section 4.2.
4. Use Nauty [25] to find the automorphism group of the graph.

The following theorem connects the Table 4.3 results to a conjecture on $\mathrm{OA}(s^t, k, s, t)$:

Theorem 14. *Let $k = i$ and assume that an $OA(\lambda s^t, k, s, t)$ exists. Let* $\operatorname{Conv}(\mathbf{M})$ *be the convex hull of the rows of* $\mathbf{M}$*. Then,* $\operatorname{rank}(\mathbf{B}) = \dim(\operatorname{Conv}(\mathbf{M}))$.

Proof. Let $\mathcal{B} = \{e_{\mathbf{x}} \mid \mathbf{x} \in \mathbf{D}_{s^k}\}$ be the standard orthonormal basis for $\mathbb{R}^{s^k}$ indexed with the full factorial design $\mathbf{D}_{s^k}$. Let $G_{s,k} = G(\mathbf{A}, \lambda\mathbf{1}) = S_{\{1,2,\cdots,s\}} \wr S_k$ act on elements of $\mathcal{B}$ by $e_{\mathbf{x}} \xrightarrow{g} e_{g\mathbf{x}}$ for each $g \in S_{\{1,2,\cdots,s\}} \wr S_k$. Let O be the orbit of $e_{\mathbf{1}}$. As the action of $S_{\{1,2,\cdots,s\}} \wr S_k$ on $\mathcal{B}$ is transitive, each element in $\mathcal{B}$ appears in O. Then, $\beta(O) = s^{-k}\mathbf{1}_{s^k}$, where $\beta(O)$ is defined as in Section 4.3. Hence, by Lemma 3 in Bödi *et al.* [4],

$$\operatorname{Fix}_{G(\mathbf{A},\lambda\mathbf{1})}\left(\mathbb{R}^{s^k}\right) = \operatorname{Span}\left(\mathbf{1}_{s^k}\right).$$

Now, let $\mathbf{y}^T_{1\times s^k}$ be a row of $\mathbf{M}$. Then, $\beta(\mathbf{y}) = \alpha\mathbf{1}_{s^k}$ for some $\alpha \in \mathbb{R}$ as β is the projection operator onto $\operatorname{Fix}_{G(\mathbf{A},\lambda\mathbf{1})}\left(\mathbb{R}^{s^k}\right)$. For $\mathbf{y} \in \mathbb{R}^{s^k}$, $S_{\{1,2,\cdots,s\}} \wr S_k$ acts on coordinates $y_{\mathbf{x}}$ of $\mathbf{y}$ by $y_{\mathbf{x}} \xrightarrow{g} y_{g^{-1}(\mathbf{x})}$. Let $O_{\mathbf{y}}$ be the orbit of $\mathbf{y}$ under this action. Then,

$$\beta(\mathbf{y}) = \sum_{\mathbf{v}\in O_{\mathbf{y}}} \frac{\mathbf{v}}{|O_{\mathbf{y}}|} = \alpha\mathbf{1}_{s^k}.$$

Each $\mathbf{v}$ satisfies the equality constraints in ILP (2.4). Then, $\sum_{\mathbf{v}\in O_{\mathbf{y}}} \frac{\mathbf{v}}{|O_{\mathbf{y}}|} = \alpha \mathbf{1}_{s^k}$ also satisfies the same constraints. This forces $\alpha = \lambda/s^{k-t}$. Hence, $(\lambda/s^{k-t})\mathbf{1}_{s^k}$ is in Conv($\mathbf{M}$). Then, $\text{rank}(\mathbf{B}) = \text{rank}\left(\mathbf{M} - \frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^k}\right) = \dim(\text{Conv}(\mathbf{M}))$. □

The second column of Table 4.3 is the size of the automorphism group, the third indicates if $|\mathcal{G}(i,s,t)| = \left|G(\mathbf{A}(i,s,t), \lambda\mathbf{1})^{\text{LP}}\right|$, the fourth is the number of rows in $\mathbf{B}^T$ $\left(= s^i\right)$, the fifth is the upper bound on the rank of $\mathbf{B}$ $\left(= \sum_{j=t+1}^{i} \binom{i}{j}(s-1)^j\right)$, the sixth is the rank of $\mathbf{B}$, and the seventh column is the minimum difference between two entries in $\mathbf{P}_{\mathbf{B}^T}$. The code used to compute this table is found in Section B.3.

Table 4.3: $\mathcal{G}(i,s,t)$ and rank($\mathbf{B}$)

OA(N,i,s,t)	$\lvert\mathcal{G}(i,s,t)\rvert$	$= \left\lvert G^{\text{LP}}\right\rvert$?	rows $\mathbf{B}^T$	UB(rank($\mathbf{B}$))	rank($\mathbf{B}$)	$\mathbf{P}_{\mathbf{B}^T}$ diff
OA$(32,6,2,4)$	322560	Y	64	7	7	0.0625
OA$(80,6,2,4)$	322560	Y	64	7	7	0.0625
OA$(64,6,2,4)$	322560	Y	64	7	7	0.0625
OA$(112,6,2,4)$	322560	Y	64	7	7	0.0625
OA$(64,7,2,4)$	5160960	Y	128	29	29	0.03125
OA$(24,7,2,3)$	5.95×10^{24}	N	128	64	42	0.03125
OA$(96,7,2,4)$	5160960	Y	128	29	29	0.03125
OA$(64,8,2,4)$	92897280	Y	256	93	84	0.015625
OA$(24,8,2,3)$	1.76×10^{45}	N	256	163	99	0.0078125
OA$(32,7,2,3)$	645120	Y	128	64	63	0.015625
OA$(24,9,2,3)$	1.08×10^{85}	N	512	382	219	1×10^{-10}
OA$(32,8,2,3)$	10321920	Y	256	163	156	1×10^{-10}

UB(rank($\mathbf{B}$)) is the dimension of the affine space to which each indicator vector of the OA$(\lambda s^t, i, s, t)$ belongs. For OA(s^t, i, s, t), Appa *et al.* [1] conjectured that UB(rank($\mathbf{B}$)) =

dim(Conv(**M**)), where Conv(**M**) is the convex hull of the rows of **M**. By Theorem 14, rank(**B**) = dim(Conv(**M**)). Hence, Table 4.3 appears to invalidate this conjecture for general OA($\lambda s^t, i, s, t$). However, there may be inaccuracies in the reported rank(**B**) values since not all solutions may have been enumerated by the solver or performing Q-R decomposition with 100 vectors at a time may have introduced numerical precision errors that lead to rejecting linearly independent rows of $\left(\mathbf{M} - \frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right)$ from **B**. If **B** is not a basis for $\left(\mathbf{M} - \frac{\lambda}{s^{k-t}}\mathbf{J}_n^{s^i}\right)$, the calculation of $\mathcal{G}(i, s, t)$ would be incorrect. Hence, it is essential to make sure that the elements in column 6 are calculated correctly. For smaller cases, in which errors are less likely, Table 4.3 not only verifies the Appa *et al.* [1] conjecture, but also suggests that $\mathcal{G}(i, s, t) = G(\mathbf{A}(i, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$.

4.5 Research Objective 4

Let $\mathbf{A}_{\mathrm{eq}}$ and $\mathbf{b}_{\mathrm{eq}}$ be the constraint matrix and the corresponding right hand side of the equality constraints in ILP (2.13). Let $G(\mathbf{A}_{\mathrm{eq}}, \mathbf{0})$ and $G(\mathbf{A}_{\mathrm{eq}}, \mathbf{b}_{\mathrm{eq}})$ be computed as described in Sections 4.2 and 4.3 respectively. Let $H^{\mathrm{LP}}_{\mathrm{eq}_1}$ and $H^{\mathrm{LP}}_{\mathrm{eq}_2}$ be the maximal subgroups of $G(\mathbf{A}_{\mathrm{eq}}, \mathbf{0})^{\mathrm{LP}}$ and $G(\mathbf{A}_{\mathrm{eq}}, \mathbf{b}_{\mathrm{eq}})^{\mathrm{LP}}$ that map the input OA($\lambda s^t, k-1, s, t$) to itself.

Theorem 15. $H^{\mathrm{LP}}_{\mathrm{eq}_1} = H^{\mathrm{LP}}_{\mathrm{eq}_2}$.

Proof. Since $G(\mathbf{A}_{\mathrm{eq}}, \mathbf{b}_{\mathrm{eq}})^{\mathrm{LP}} \subseteq G(\mathbf{A}_{\mathrm{eq}}, \mathbf{0})^{\mathrm{LP}}$, then $H^{\mathrm{LP}}_{\mathrm{eq}_2} \subseteq H^{\mathrm{LP}}_{\mathrm{eq}_1}$. Let $\hat{\mathbf{x}}$ be such that

$$\hat{x}_{(i_l-1)s+j} = \frac{r_{i_l}}{s}$$

where $j = 1, 2, \cdots, s-1$, and i_l be as in the modified ILP (2.13). Then, $\hat{\mathbf{x}}$ is a fractional solution to the modified ILP (2.13). Hence, every solution to the LP relaxation of the modified ILP (2.13) can be written in the form

$$p_{\max}\mathbf{1} \geq \mathbf{x} = \hat{\mathbf{x}} + \mathbf{v} \geq 0$$

for some $\mathbf{v}$ such that $-\hat{\mathbf{x}} \leq \mathbf{v} \leq p_{\max}\mathbf{1} - \hat{\mathbf{x}}$ and $\mathbf{v} \in \mathrm{Null}(\mathbf{A}_{\mathrm{eq}})$. This is true because $\hat{\mathbf{x}}$ is a particular solution to $\mathbf{A}_{\mathrm{eq}}\mathbf{x} = \mathbf{b}_{\mathrm{eq}}$ and $\mathbf{x} \geq \mathbf{0}$. Now, the set of all permutations

of variables that map the input $\text{OA}(\lambda s^t, k-1, s, t)$ to itself must map $\hat{\mathbf{x}}$ to itself. Then, $G(\mathbf{A}_{\text{eq}}, \mathbf{b}_{\text{eq}})^{\text{LP}} \cap H^{\text{LP}}_{\text{eq}_2} = G(\mathbf{A}_{\text{eq}}, \mathbf{0})^{\text{LP}} \cap H^{\text{LP}}_{\text{eq}_1}$. Thus, $H^{\text{LP}}_{\text{eq}_1} = H^{\text{LP}}_{\text{eq}_2}$. □

Add back the variables $\{x_{(i_l-1)s+s}\}_{l=1}^{h}$ to ILP (2.13), and replace $\sum_{j=1}^{s-1} x_{(i_l-1)s+j} \leq r_{i_l}$ with $\sum_{j=1}^{s} x_{(i_l-1)s+j} = r_{i_l}$. Also, add the constraints $\sum_{l=1}^{h} x_{(i_l-1)s+s} = \lambda s^{t-1}$ and $\sum_{l=1}^{h} d'_{p_l\alpha_1} d'_{p_l\alpha_2} \cdots d'_{p_l\alpha_{q-1}} x_{(i_l-1)s+s} = \lambda s^{t-q}$. Let $\mathbf{A}_{\text{all}}$ be the resulting matrix whose rows correspond to the equality constraints in the modified ILP (2.13), and $\mathbf{b}_{\text{all}}$ be the corresponding right hand side. Let $G(\mathbf{A}_{\text{all}}, \mathbf{0})^{\text{LP}}$ and $G(\mathbf{A}_{\text{all}}, \mathbf{b}_{\text{all}})^{\text{LP}}$ be computed as described in Sections 4.2 and 4.3 respectively. Let $H^{\text{LP}}_{\text{all}_1}$ and $H^{\text{LP}}_{\text{all}_2}$ be the maximal subgroups of $G(\mathbf{A}_{\text{all}}, \mathbf{0})^{\text{LP}}$ and $G(\mathbf{A}_{\text{all}}, \mathbf{b}_{\text{all}})^{\text{LP}}$ that map the input $\text{OA}(\lambda s^t, k-1, s, t)$ to itself.

Theorem 16. $H^{\text{LP}}_{\text{all}_1} = H^{\text{LP}}_{\text{all}_2}$.

The proof is skipped as it is analogous to the proof of Theorem 15.

Let $H_{s,k}$ be the maximal subgroup of $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})$ (as defined in Section 2.4) that maps the input $\text{OA}(\lambda s^t, k-1, s, t)$ to itself and preserves $\{x_{(i_l-1)s+s} : l = 1, 2 \cdots, h\}$. Let $H^{\text{LP}}_{s,k}$ be the maximal subgroup of $G(\mathbf{A}(k, s, t), \lambda\mathbf{1})^{\text{LP}}$ from Section 4.2 that maps the input $\text{OA}(\lambda s^t, k-1, s, t)$ to itself and preserves $\{x_{(i_l-1)s+s} : l = 1, 2, \cdots, h\}$. When $H^{\text{LP}}_{s,k}$ or $H^{\text{LP}}_{\text{all}_1}$ is used, some OD classes of $\text{OA}(\lambda s^t, k, s, t)$ may be lost. However, at least one $\text{OA}(\lambda s^t, k, s, t)$ will be found if the input $\text{OA}(\lambda s^t, k-1, s, t)$ can be extended to an $\text{OA}(\lambda s^t, k, s, t)$.

The Figure 2.6 algorithm was used to enumerate all non-OD equivalent $\text{OD}(160, k, 2, 4)$ for $k \leq k_{\max}(160, 2, 4)$ by solving ILPs (2.13) and modified ILPs (2.13). The modified ILPs (2.13) were solved using $H^{\text{LP}}_{\text{all}_1}$ and $H^{\text{LP}}_{s,k}$. $H_{s,k}$ and $H^{\text{LP}}_{\text{eq}_1}$ were used to solve the original ILPs (2.13). Comparisons were made for efficiency. Only using $H_{s,k}$ or $H^{\text{LP}}_{\text{eq}_1}$ guarantee that no OD classes are lost. However, a comparison of the found number of non-OD equivalent OAs to those found using $H_{s,k}$ shows that none of the methods lost OD classes of OAs. At the end of each extension step, right before the Nauty [25] reduction, using $H_{s,k}$ and $H^{\text{LP}}_{\text{eq}_1}$ produced the exact same number of solutions. In three of the extensions to $k = 8$, $H^{\text{LP}}_{\text{eq}_1}$ was in the order of 10^{35}. These extensions were the only infeasibles. For the remaining feasible

extensions, $\left|H^{\text{LP}}_{\text{eq}_1}\right| = \left|H_{s,k}\right|$. Also, for each extension to $k = 9, 10$, $\left|H^{\text{LP}}_{\text{eq}_1}\right| = \left|H_{s,k}\right|$. Hence, using $H^{\text{LP}}_{\text{eq}_1}$ was slightly slower than using $H_{s,k}$ as it requires calculating the projection matrix $\mathbf{P}_{\mathbf{A}^T_{\text{eq}}}$ for each extension. In the extensions to $k = 11$, $\left|H^{\text{LP}}_{\text{eq}_1}\right| = \left|H_{s,k}\right|$ in all but four of the extensions. In those four extensions, $\left|H^{\text{LP}}_{\text{eq}_1}\right|$ is much larger than $\left|H_{s,k}\right|$. This explains why using $H^{\text{LP}}_{\text{eq}_1}$ was about seven times faster than using $H_{s,k}$ in the extension to $k = 11$. However, using $H_{s,k}$ was still slightly faster for the whole $OA(160, k, 2, 4)$ enumeration starting at $k = 7$.

Using $H^{\text{LP}}_{s,k}$ and $H^{\text{LP}}_{\text{all}_1}$ resulted in similar solution times. Using $H^{\text{LP}}_{\text{all}_1}$ outperformed $H_{s,k}$ for extensions up to eight factors. However, $H_{s,k}$ was much faster than $H^{\text{LP}}_{\text{all}_1}$ for the extension from eight factors to nine factors. Since this is the bottleneck extension, overall, using $H_{s,k}$ was faster than using $H^{\text{LP}}_{\text{all}_1}$ or $H^{\text{LP}}_{s,k}$. The number of non-OD equivalent OAs found by each extension method and their respective run times are listed in Table 4.4.

OA$(176, k, 2, 4)$ enumerations were also made to asses the speed of solving modified ILPs (2.13) with $H^{\text{LP}}_{\text{all}_1}$. The same sequence of enumerations will be implemented by solving ILPs (2.13) with $H_{s,k}$. The increasing computational time in solving modified ILPs (2.13) with $H^{\text{LP}}_{\text{all}_1}$ for the OA$(176, k, 2, 4)$ cases suggests that the OA$(192, k, 2, 4)$ cases are out of computational reach even if ILPs (2.13) are solved with $H_{s,k}$.

Table 4.4: Method Comparisons

OA(N, k, s, t)	# of Non-OD Equivalent OAs				Times (sec.)			
	$H^{\mathrm{LP}}_{s,k}$	$H^{\mathrm{LP}}_{\mathrm{all}_1}$	$H_{s,k}$	$H^{\mathrm{LP}}_{\mathrm{eq}_1}$	$H^{\mathrm{LP}}_{s,k}$	$H^{\mathrm{LP}}_{\mathrm{all}_1}$	$H_{s,k}$	$H^{\mathrm{LP}}_{\mathrm{eq}_1}$
OA(160,7,2,4)	106	106	106	106	339.85	443.37	810.73	881.41
OA(160,8,2,4)	11712	11712	11712	11712	71410	72433	82800	92509
OA(160,9,2,4)	1608	1608	1608	1608	598324	629048	412753	473562
OA(160,10,2,4)	0	0	0	0	43299	17295	73159	10333
OA(176,7,2,4)	-	179	-	-	-	917	-	-
OA(176,8,2,4)	-	129138	-	-	-	1134186	-	-
OA(176,9,2,4)	-	4	-	-	-	19313974	-	-
OA(176,10,2,4)	-	0	-	-	-	4	-	-

The code to compute $H_{s,k}$ is found in Bulutoglu and Ryan [8]. The modification to the Bulutoglu and Ryan [8] code to make it use $H^{\mathrm{LP}}_{s,k}$ is found in Section B.4. The modification to the Bulutoglu and Ryan [8] code to make it use $H^{\mathrm{LP}}_{\mathrm{all}_1}$ is found in Section B.5.

V. Conclusions and Future Research

5.1 Conclusions

In Chapter 3, it was first shown that none of the inequalities in the LP relaxation of ILP (3.2) that describes OA(N, k, s, t)s is redundant. This renders each inequality to be a facet of the LP relaxation. In Section 3.3, this result was used to explicitly show that the symmetry group $G(\mathbf{A}'(k, s, t), \mathbf{b}_m)^{\mathrm{LP}}$ of the LP relaxation of ILP (3.2) is $S_{\{2,\cdots,s\}} \wr S_k$. Even though it is easy to see that $S_{\{2,\cdots,s\}} \wr S_k \subseteq G(\mathbf{A}'(k, s, t), \mathbf{b}_m)^{\mathrm{LP}}$, it is far from trivial to show that $S_{\{2,\cdots,s\}} \wr S_k = G(\mathbf{A}'(k, s, t), \mathbf{b}_m)$. The practical value of this result is that there are no additional symmetries that can be exploited by using a subgroup of $G(\mathbf{A}'(k, s, t), \mathbf{b}_m)^{\mathrm{LP}}$ to find OAs by solving ILP (3.2).

Bulutoglu and Margot [7] previously proved that $G_{s,k} = S_{\{1,\cdots,s\}} \wr S_k$ is a subgroup of the automorphism group $G(\mathbf{A}, \lambda\mathbf{1}, \mathbf{1}, p_{\max}\mathbf{1})$ of ILP (2.4). Based on computational observations, they conjectured that $G_{s,k} = G(\mathbf{A}, \lambda\mathbf{1}, \mathbf{1}, p_{\max}\mathbf{1})$. This conjecture was proved in Chapter 3.

In Section 4.2.1, the Margot ILP solver [22] was used to find a set of all non-isomorphic solutions to ILP (3.2) exploiting $G(\mathbf{A}(k, s, t), \mathbf{b}_m)^{\mathrm{LP}}$. This was compared to finding all non-isomorphic solutions to ILP (2.5), which has more variables, by exploiting the larger group $G_{s,k}$. It is legitimate to use $G_{s,k}$ to solve ILP (2.5) as ILP (2.5) and ILP (2.4) have the same feasible sets defined by the same variables. For most of the cases, a set of all non-isomorphic solutions to ILP (2.5) with the larger number of variables was found faster.

When the Margot ILP solver [22] is used to solve a symmetric ILP, it is essential to find larger subgroups of the ILP's symmetry group to speed up the enumeration. One easy-to-compute subgroup is the automorphism group of the formulation, $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$. This group coincides with the symmetry group of the LP relaxation $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\mathrm{LP}}$, provided that the LP relaxation has no redundant constraints and is full dimensional. Hence, given an ILP formulation whose LP relaxation is full dimensional, finding $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\mathrm{LP}}$ is a matter of

culling the redundant constraints and computing the automorphism group of the resulting formulation. On the other hand, for an ILP whose LP relaxation is not full dimensional, the LP relaxation may contain hidden symmetries not captured by $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$, even after removing all the redundant constraints.

The LP relaxation symmetry group always contains the automorphism group of an ILP formulation. Thus, it is more desireable to exploit the LP relaxation symmetry group. The LP relaxation of an ILP with equality constraints is not full dimensional. Hence, it is possible that the LP relaxation symmetry group of such an ILP strictly contains the ILP formulation automorphism group. In Sections 4.2 and 4.3, an algorithm was developed for finding this potentially larger group. This is the first algorithm for this purpose in the literature.

In Section 4.2, a special case of the developed algorithm is applied to LP relaxations of ILPs with equality constraints that describe orthogonal arrays. Computational results not only revealed previously unknown hidden symmetries of the ILP (2.5) formulation of the OA problem, but also demonstrated that exploiting the newly found symmetries decreases solution times significantly. In Section 4.2.2, these newly found hidden symmetries were explicitly described as the automorphism group $G^J_{k,t}$ of a system of equations derived from equations (4.4). Furthermore, for each k, t combination, $G^J_{k,t}$ was conjectured to be the same as the symmetry group of the LP relaxation of ILP (2.5).

In Section 4.3, the newly developed algorithm was used to find the LP relaxation symmetry groups of modified versions of MILPs featured in Liberti [18]. All inequalities in each MILP were first converted to equality constraints by adding slack variables. In many of these cases,

$$|G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})^{\mathrm{LP}}| > |G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})|.$$

Hence, this algorithm reveals hidden symmetries that do not exist in $G(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{d})$. If a MILP solver with isomorphism pruning is used on these problems, it is expected that exploiting

the larger groups would overcome the additional computational burden due to the added slack variables.

In Section 4.4, $\mathcal{G}(k, s, t)$, the symmetry group of ILP (2.5), was calculated for several k, s, t combinations. For most cases, all the symmetry in the OA problem was captured by the symmetry group of its LP relaxation. Differences between these groups occurred in larger cases where numerical precision errors were more likely to throw off calculations. As a side benefit, the method developed for calculating $\mathcal{G}(k, s, t)$ also checked the validity of a conjecture made by Appa *et al.* [1]. The fact that this conjecture was verified only in the smaller cases suggests that this method suffers from numerical precision errors for the larger cases.

In Section 4.5, the LP relaxation symmetry group was used in various ways for each of the OA extensions based on ILPs (2.13) in the Bulutoglu and Ryan [8] OA extension algorithm. Even though there were some extensions in which larger groups were found, the computational savings did not make up for the time lost finding these groups. The OA extension algorithm developed by Bulutoglu and Ryan [8] still remains the fastest known for enumerating all non-isomorphic OA$(160, k, 2, 4)$ and OA$(176, k, 2, 4)$.

In the next section, some open problems are presented that arose from the research in this dissertation. In Section 5.3, a set of conjectures is provided based on observations from the computational research in Chapter 4. It is proposed that future research efforts resolve these conjectures by either proving them or finding counterexamples. In Section 5.4, a list of computational projects is proposed to improve upon the computional research in Chapter 4.

5.2 Open Problems

Problem 1. *Determine the isomorphism class of the group* $G(\boldsymbol{A}(k, s, t), \lambda\mathbf{1})^{\mathrm{LP}}$ *in Section 4.1 by determing how it relates to its subgroup* $G(\boldsymbol{A}(k, s, t), \lambda\mathbf{1}) = S_{\{1,2,\cdots,s\}} \wr S_k$.

Problem 2. *Develop the theory and computational tools in this dissertation for t-designs.*

5.3 Conjectures

Conjecture 2. *If an OA($\lambda s^t, i, s, t$) exists, then dim($Conv(\mathbf{M})$) = $\sum_{j=t+1}^{i} \binom{i}{j}(s-1)^j$, where* $\mathbf{M}$ *is as in Section 4.4.*

Conjecture 3. *If an OA($\lambda s^t, i, s, t$) exists, then $\mathcal{G}(i,s,t) = G(\boldsymbol{A}(i,s,t), \lambda\mathbf{1})^{\mathrm{LP}}$, where $G(\boldsymbol{A}(i,s,t), \lambda\mathbf{1})^{\mathrm{LP}}$ is as in Section 4.2 and $\mathcal{G}(i,s,t)$ is as in Section 4.4.*

Conjecture 4. *Let $G(\boldsymbol{A}(k,s,t), \lambda\mathbf{1})^{\mathrm{LP}}$ be as in Section 4.1. Then,*

$$\left|G(\boldsymbol{A}(k,s,t), \lambda\mathbf{1})^{\mathrm{LP}}\right| = \begin{cases} (k+1)!2^k & \text{if } t \text{ is even, } s = 2, \text{ and } 1 \le t < k, \\ k!(s!)^k & \text{if } (t \text{ isodd or } s > 2) \text{ and } 1 \le t < k, \\ s^k! & \text{otherwise.} \end{cases}$$

To prove this conjecture, let $\boldsymbol{E}(N,k,t)$ be the constraint matrix of the system of equations (4.4). Now, let $\boldsymbol{P}_{\boldsymbol{E}(N,k,t)^T}$ be the projection matrix onto the row space of $\boldsymbol{E}(N,k,t)$. Then, as the rows of $\boldsymbol{E}(N,k,t)$ are orthogonal, $\boldsymbol{P}_{\boldsymbol{E}(N,k,t)^T} = \frac{1}{N}\boldsymbol{E}(N,k,t)^T\boldsymbol{E}(N,k,t)$. Furthermore, $G(\boldsymbol{A}(k,2,t), \lambda\mathbf{1})^{\mathrm{LP}}$ is the automorphism group of this matrix. After fixing an order for the variables, come up with a closed form formula for $\boldsymbol{P}_{\boldsymbol{E}(N,k,t)^T}$. Use this formula to calculate the automorphism group.

Conjecture 5. *Let $\boldsymbol{A}_{\mathrm{all}}$, $\boldsymbol{b}_{\mathrm{all}}$, and $H^{\mathrm{LP}}_{s,k}$ be as in Section 4.5. Let $\hat{\boldsymbol{A}}_{\mathrm{all}}$ and $\hat{\boldsymbol{b}}_{\mathrm{all}}$ be obtained from $\boldsymbol{A}_{\mathrm{all}}$ and $\boldsymbol{b}_{\mathrm{all}}$ by deleting the constraints $\sum_{j=1}^{s} x_{(i_l-1)s+j} = r_{i_l}$. Let $G(\hat{\boldsymbol{A}}_{\mathrm{all}}, \mathbf{0})^{\mathrm{LP}}$ and $G(\hat{\boldsymbol{A}}_{\mathrm{all}}, \hat{\boldsymbol{b}}_{\mathrm{all}})^{\mathrm{LP}}$ be computed as described in Sections 4.2 and 4.3, respectively. Let $\hat{H}^{\mathrm{LP}}_{\mathrm{all}_1}$ and $\hat{H}^{\mathrm{LP}}_{\mathrm{all}_2}$ be the maximal subgroups of $G(\hat{\boldsymbol{A}}_{\mathrm{all}}, \mathbf{0})^{\mathrm{LP}}$ and $G(\hat{\boldsymbol{A}}_{\mathrm{all}}, \hat{\boldsymbol{b}}_{\mathrm{all}})^{\mathrm{LP}}$ that map the input OA($\lambda s^t, k-1, s, t$) to itself. Then, $\hat{H}^{\mathrm{LP}}_{\mathrm{all}_1} = \hat{H}^{\mathrm{LP}}_{\mathrm{all}_2} = H^{\mathrm{LP}}_{s,k}$.*

5.4 Computational Research Improvements

Project 1. *Resolve the numerical precision issues in the methods and algorithms used to develop Table 4.3.*

Project 2. *If Conjecture 5 is proven, compute* $\hat{H}^{\mathrm{LP}}_{\mathrm{all}_1}$ *by calculating the singular value decomposition of* $\hat{\mathbf{A}}_{\mathrm{all}}$ *as described in Section 4.2 for all of the algorithms in this dissertation that require* $H^{\mathrm{LP}}_{s,k}$.

Appendix A: Computer Code for Theoretical Research

A.1 MATLAB Code for ILP (2.4) Constraints

Appendix1/Constraintoa.m

```
function A = Constraintoa(k,s,t)
% Usage:  A<-constraintoa(8,2,2)

   filePath = mfilename('fullpath');
   filePath = filePath(1:(length(filePath)-length(mfilename(''))));
   [~, ~] = system(strcat('mkdir',32,filePath,'A_Matrices'));
   filePath = ...
      strcat(filePath,'A_Matrices/Ak',num2str(k),'s',num2str(s),...
       't',num2str(t),'.txt');

   % Vector of 1's of length s
   ps = ones(1,s);
   % Identity matrix of size s
   iss = eye(s);
   % Generate all k choose t combinations
   indexmat = combntns(1:k,t);

   % Count the number of cases
   numcases = size(indexmat, 1);

   % Initialize A with an empty matrix
   %A=[];

   fcoa = fopen(filePath,'w');

   % Build the A matrix
```

```
    for j = 1:1:numcases
        temp = 1;
        for i = 1:1:k
            if sum(i*ones(1,t)==indexmat(j,:))==1
                temp = kron(temp,iss);
            else
                temp = kron(temp,ps);
            end;
        end;
        %A = [A; temp];
        for i = 1:1:size(temp,1)
        fprintf(fcoa,strcat(repmat('%d ...
            ',1,size(temp,2)-1),32,'%d\n'),temp(i,:));
        end;
    end;

    fclose(fcoa);

    A = dlmread(filePath);
```

A.2 Column Permutations of ILP (2.4) Constraint Matrix

Appendix1/kstRandPermProb.m

```
clear; clc;

%----Determine folder containing the m file
filePath = mfilename('fullpath');
filePath = filePath(1:(length(filePath)-length(mfilename(''))));

%----List of fraction probabilities----
fileName = strcat(filePath,'FractionProb.csv');
fprob = fopen(fileName,'w');
```

```
fprintf(fprob,strcat('k,s,t,# Frac,# Total,P(Frac)\n'));
fclose(fprob);

for s = 2:1:4
    for k = 4:1:round(26/s)
        for t = 1:1:(k-2)

            X = [k s t zeros(1,3)];

            %----Display current OA to user----
            disp(strcat('Processing OA(k=',num2str(k),', ...
                s=',num2str(s), ...
                ', t=', num2str(t),')....'));

            %----Constraint OA matrix----
            A = Constraintoa(k,s,t);
            n = size(A,2);
            X(5) = 1000;
            for i = 1:1:X(5)
                R = A(:,randperm(n));
                R = [R ones(size(R,1),1)];
                R = rref(R);
                if sum(sum((R-round(R))>0))>0
                   X(4) = X(4) + 1;
                end;
            end;
            clear A R;
            X(6) = X(4) / X(5);

            dlmwrite(fileName,X,'-append','delimiter',',');
        end;
    end;
```

```
41 end;
```

A.3 MATLAB Code to Generate Rosenberg [29] Constraints

Appendix1/Rosenberg.m

```
1 function [a, A, rhs] = Rosenberg(k, s, t, lam)
2 %UNTITLED Summary of this function goes here
3 %   Detailed explanation goes here
4
5 %Rosenberg=function(k,s,t,lam){
6 %a=as.matrix(c(rep(0,(t+1))))
7 a = zeros(t + 1, 1);
8 a(1,1) = lam;
9 cc = 1;
10 while cc < t + 1
11     tut = lam;
12     for i = 1:1:cc
13         if (cc-i+1) <= (k-t)
14             tut = tut-a(i,1)*nchoosek((k-t),(cc-i+1))*(s-1)^(cc-i+1);
15         end;
16     end;
17     a(cc + 1, 1) = tut;
18     cc = cc + 1;
19 end;
20
21 N = 0;
22 for i = 0:1:t
23     N = N + nchoosek(k, i) * (s - 1)^i;
24 end;
25 m = s^k - N;
26 A = zeros(N,m);
27 rhs = zeros(N,1);
```

```
r1 = 1;
for j = 1:1:(s^k)
    mm1 = dec2base(j-1, s, 32)-'0';
    m1 = sum(mm1>0);
    x = find(mm1>=1);
    if m1 <= t
        ccc = t - m1;
        r2 = 1;
        for i = 1:1:(s^k)
            mm2 = dec2base(i-1, s, 32)-'0';
            m2 = sum(mm2>0);
            if m2 > t
                if sum((mm2(x)-mm1(x)).^2>0) == 0 && ccc <= m2-t+cc-1
                    A(r1,r2) = (-1)^(ccc+1)*nchoosek(m2-t+ccc-1,ccc);
                end;
                r2 = r2 + 1;
            end;
        end;
        rhs(r1,1) = -1*a(ccc+1,1);
        r1 = r1 + 1;
    end;
end;

end
```

A.4 MATLAB Code to Test Theorem 6

Appendix1/newFacetTest.m

```
clear; clc;

%if matlabpool('size') == 0
%    matlabpool open local 3
```

```
5 %end;
6
7 options=optimset('MaxIter', 100000, 'TolFun', 10e-8);
8 filePath = strrep(mfilename('fullpath'),mfilename(''),'');
9 addpath(filePath);
10 cmd_line = strcat('mkdir',32,filePath,'Facet_Solutions');
11 [~,~] = system(cmd_line);
12 filePath = strcat(filePath,'Facet_Solutions/');
13
14  Cases=[6,2,3,5
15         5,2,2,1
16         5,2,2,2
17         5,3,2,2
18         5,3,3,2
19         5,3,3,3
20         5,3,2,3
21         4,2,2,1
22         8,2,2,2
23         7,2,2,1
24         6,2,2,1
25         9,2,2,2
26         7,2,2,2
27         6,2,2,2
28         7,2,3,7
29         8,2,4,5
30         9,2,4,5
31         9,2,5,5
32         9,2,3,6
33         10,2,4,11
34         10,2,5,12
35         6,3,3,2
36         7,3,3,7
```

```
        8,3,3,7
        9,3,2,2];

CaseCount = size(Cases,1);
k = Cases(:,1);
s = Cases(:,2);
t = Cases(:,3);
l = Cases(:,4);

for i=1:1:CaseCount
    ko = k(i);
    so = s(i);
    to = t(i);
    lo = l(i);
    [~,A,b]=Rosenberg(ko,so,to,lo);
    n=size(A,2);
    b = -1*[b;zeros(n,1)];
    A = -1*[A;eye(n)];
    m=size(A,1);
    fileName1 = ...
        strcat(filePath,'edgeXk',num2str(ko),'s',num2str(so),'t', ...
        num2str(to),'lam',num2str(lo),'.csv');
    fileName2 = ...
        strcat(filePath,'cornerXk',num2str(ko),'s',num2str(so),'t', ...
        num2str(to),'lam',num2str(lo),'.csv');
    for j = 0:1:((ko+1)^n-1) %#ok<PFRNG>
       Xo = (dec2base(j,ko+1,n)-'0')';
       d = b - A*((lo/so^(ko-to))*Xo);
       if min(d) >= 0 && sum(d==0) > 0
           display(sum(d==0));
           display(Xo');
           if sum(d==0) == 1
```

```
67                 fileName = fileName1;
68             else
69                 fileName = fileName2;
70             end;
71             dlmwrite(fileName,Xo','-append','delimiter',',');
72         end;
73     end;
74 end;
75
76 %matlabpool close;
```

A.5 MATLAB Code to Implement Espinoza [9]

Appendix1/ExactSolve.m

```
1  function [ status, X ] = ExactSolve( f, A, b, Aeq, beq, lb, ub, ...
       FileName )
2  %UNTITLED2 Summary of this function goes here
3  %   Detailed explanation goes here
4
5  n = max([size(A,2), size(Aeq,2)]);
6  X = zeros(n,1);
7
8  FilePath = strrep(mfilename('fullpath'),mfilename(''),'');
9  addpath(strcat(FilePath,'BuildMPS'));
10 InputPath = strcat('mkdir',32,FilePath,'MPS_Files');
11 [~,~] = system(InputPath);
12 InputPath = strcat(FilePath,'MPS_Files');
13 OutputPath = strcat('mkdir',32,FilePath,'Solver_Output');
14 [~,~] = system(OutputPath);
15 OutputPath = strcat(FilePath,'Solver_Output');
16 addpath(strcat(FilePath,'Projection_Matrices'));
17
```

```
18 InputPath = strcat(InputPath,'/',FileName,'.mps');
19 OutputPath = strcat(OutputPath,'/',FileName,'.sol');
20
21 [~, OK]=BuildMPS(A, b, Aeq, beq, f, lb, ub,'INPUT', ...
22     'MPSfilename',strcat('MPS_Files/',FileName,'.mps'));
23
24 if OK == 0
25     return;
26 end;
27
28 cmd_line = strcat(FilePath,'QSopt_ex-2.5.10/bin/./mpq_solver -O',32, ...
29     InputPath,32,'>',32,OutputPath);
30
31 [status, ~] = system(cmd_line);
32
33 if status == 0
34     fin = fopen(OutputPath);
35     tline = fgets(fin);
36     while ischar(tline) && length(tline) == ...
37             length(strrep(tline,'Solution Values',''))
38         tline = fgets(fin);
39     end;
40     if length(tline) > length(strrep(tline,'Solution Values',''))
41         while ischar(tline) && length(tline) == ...
               length(strrep(tline,'=',''))
42             tline = fgets(fin);
43         end;
44         while ischar(tline)
45             if length(tline) > length(strrep(tline,'=',''))
46                 i = strfind(tline, '=')-1;
47                 j = str2double(strrep(tline(1:i),'X',''));
48                 k = sym(tline((i+2:length(tline))));
```

```
49                  X(j,1) = k;
50              end;
51              tline = fgets(fin);
52          end;
53      else
54          status = 2;
55      end;
56      fclose(fin);
57  end;
58
59  end
```

A.6 MATLAB Code to Remove Layers in Nauty [25] Output

Appendix1/GLPRemoveLayers.m

```
1  function GLPRemoveLayers(dfix, numvar, infile, outfile)
2  % This function removes layers for the GLP group
3
4  fin = fopen(infile);
5  fout = fopen(outfile, 'w');
6
7  tline = fgets(fin);
8  A = [];
9  linelen = 78;
10
11 while ischar(tline)
12     % set correct number of vertices
13     if length(tline) > length(strrep(tline,'n=',''))
14         j = 1;
15         while j < (length(tline)-8) && ...
16                 ~strcmp(tline(j:(j + 1)),'n=')
17             j = j + 1;
```

```
18          end;
19          j = j + 2;
20          q = j;
21          while q < length(tline) && ~strcmp(tline(q),' ')
22              q = q + 1;
23          end;
24          n = str2double(tline(j:q));
25          tline = strrep(tline,strcat('n=', num2str(n)), ...
                strcat('n=',num2str(numvar)));
26      end;
27      if length(tline) > length(strrep(tline,'linelen=',''))
28          j = 1;
29          while j < (length(tline)-8) && ...
30                  ~strcmp(tline(j:(j + 7)),'linelen=')
31              j = j + 1;
32          end;
33          j = j + 8;
34          q = j;
35          while q < length(tline) && ~strcmp(tline(q),' ')
36              q = q + 1;
37          end;
38          n = str2double(tline(j:q));
39          tline = strrep(tline,strcat('linelen=', num2str(n)), ...
                strcat('linelen=',num2str(linelen)));
40      end;
41      B = str2num(tline); %#ok<ST2NM>
42      if max(size(B)) == 0 || ~strcmp(tline(1:4),'    ')
43          if max(size(A)) > 0
44              A = floor(A./dfix);
45              [~,index] = unique(A,'first'); % Capture the index, ...
                    ignore junk
```

```
46            A = A(sort(index));                % Index A with the ...
                 sorted index
47            k = max(size(A));
48            outtext = ' ';
49            for i=1:1:k
50                if length(outtext) + length(num2str(A(i))) < linelen
51                    outtext = strcat(outtext,32,num2str(A(i)));
52                else
53                    fprintf(fout,strcat(outtext,'\n'));
54                    outtext = strcat(32,32,32,32,num2str(A(i)));
55                end;
56            end;
57            fprintf(fout,strcat(outtext,'\n'));
58            A = [];
59         end;
60         if max(size(B)) == 0
61            fprintf(fout,strcat(tline,'\n'));
62         else
63             A = B;
64         end;
65     else
66         A = [A, B]; %#ok<AGROW>
67     end;
68 %    if length(tline) > length(strrep(tline,'linelen','',''))
69 %        j = 1;
70 %        while j < (length(tline)-8) && ...
71 %                ~strcmp(tline(j:(j + 7)),'linelen=')
72 %            j = j + 1;
73 %        end;
74 %        j = j + 8;
75 %        q = j;
76 %        while q < length(tline) && ~strcmp(tline(q),' ')
```

```
77 %             q = q + 1;
78 %         end;
79 %         linelen = str2double(tline(j:q));
80 %     end;
81     tline = fgets(fin);
82 end;
83
84 fclose(fin);
85 fclose(fout);
86
87 end
```

A.7 MATLAB Code to Test Theorem 8

Appendix1/ProjMatrix.m

```
1 clear; clc;
2
3 %----Determine folder containing the m file
4 filePath = mfilename('fullpath');
5 filePath = filePath(1:(length(filePath)-length(mfilename(''))));
6 Linux_Command = strcat(filePath,'Linux_Command.txt');
7 fcom = fopen(Linux_Command,'w');
8
9 %----Make subdirectories for files----
10 [~,~] = system(strcat('mkdir',32,filePath,'Projection_Matrices'));
11 addpath(strcat(filePath,'Projection_Matrices'));
12 [~,~] = system(strcat('mkdir',32,filePath,'g6_Files'));
13 addpath(strcat(filePath,'g6_Files'));
14 [~,~] = system(strcat('mkdir',32,filePath,'Junk'));
15 addpath(strcat(filePath,'Junk'));
16 [~,~] = system(strcat('mkdir',32,filePath,'Gap_Input'));
17 addpath(strcat(filePath,'Gap_Input'));
```

```
[~,~] = system(strcat('mkdir',32,filePath,'Dreadnaut_Input'));
addpath(strcat(filePath,'Dreadnaut_Input'));
[~,~] = system(strcat('mkdir',32,filePath, ...
    'Dreadnaut_Input/Permutation'));
addpath(strcat(filePath,'Dreadnaut_Input/Permutation'));
[~,~] = system(strcat('mkdir',32,filePath,'Dreadnaut_Input/Cycle'));
addpath(strcat(filePath,'Dreadnaut_Input/Cycle'));
[~,~] = system(strcat('mkdir',32,filePath,'Dreadnaut_Output'));
addpath(strcat(filePath,'Dreadnaut_Output'));
[~,~] = system(strcat('mkdir',32,filePath, ...
    'Dreadnaut_Output/Original'));
addpath(strcat(filePath,'Dreadnaut_Output/Original'));
[~,~] = system(strcat('mkdir',32,filePath, ...
    'Dreadnaut_Output/Original/Permutation'));
addpath(strcat(filePath,'Dreadnaut_Output/Original/Permutation'));
[~,~] = system(strcat('mkdir',32,filePath, ...
    'Dreadnaut_Output/Original/Cycle'));
addpath(strcat(filePath,'Dreadnaut_Output/Original/Cycle'));
[~,~] = system(strcat('mkdir',32,filePath,'Dreadnaut_Output/Reduced'));
addpath(strcat(filePath,'Dreadnaut_Output/Reduced'));
[~,~] = system(strcat('mkdir',32,filePath, ...
    'Dreadnaut_Output/Reduced/Permutation'));
addpath(strcat(filePath,'Dreadnaut_Output/Reduced/Permutation'));
[~,~] = system(strcat('mkdir',32,filePath, ...
    'Dreadnaut_Output/Reduced/Cycle'));
addpath(strcat(filePath,'Dreadnaut_Output/Reduced/Cycle'));

%----Linux command to run gap silently----
gap = '/usr/share/gap4r5/bin/gap-default64.sh -q';

%----Linux command line script for Hnosigneff executable----
Hnosigneff = strcat(filePath,'Hnosigneff');
```

```
50
51 %----Nauty Linux command line scripts----
52 nautyFile = strcat(filePath,'nauty25rc1/');       % Folder ...
       containing nauty
53 listg = strcat(nautyFile,'listg -d');             % listg executable
54 dreadnaut = strcat(nautyFile,'dreadnaut <');      % dreadnaut executable
55
56 %----List of permutation group sizes----
57 ftime = fopen(strcat(filePath,'Run_Times.csv'),'w');
58 fprintf(ftime,strcat('k,s,t,Constraintoa,', ...
59     'Projection Matrix,Lower Half of P,Hnosigneff,listg,dreadnaut,', ...
60     'RemoveLayers,Gap\n'));
61 fsizes = fopen(strcat(filePath,'Group_Sizes.csv'),'w');
62 fprintf(fsizes,strcat('k,s,t,dfix,k!(((s-1)!)^k,Stabilizer,', ...
63     '(k+1)k!(s!)^k,Group Size\n'));
64
65 for k = 4:1:7
66     for s = 2:1:4
67         for t = 2:1:min([(k-1) 4])
68             %----Display current OA to user----
69             disp(strcat('Processing OA(k=',num2str(k),', ...
                   s=',num2str(s), ...
70                  ', t=', num2str(t),')....'));
71             fileLabel = strcat('k',num2str(k),'s',num2str(s),'t', ...
72                 num2str(t));
73             PfileName = strcat(filePath,'Projection_Matrices/P', ...
74                 fileLabel,'.txt');
75             g6fileName = strcat(filePath,'g6_Files/P',fileLabel,'.g6');
76             dreadInPerm = strcat(filePath, ...
77                 'Dreadnaut_Input/Permutation/aa',fileLabel,'.naut.inp');
78             dreadInCycle = ...
                   strcat(filePath,'Dreadnaut_Input/Cycle/aa', ...
```

```
79              fileLabel,'.naut.inp');
80          dreadOutPerm = strcat(filePath, ...
81            'Dreadnaut_Output/Original/Permutation/aa',fileLabel,'.out');
82          dreadOutCycle = strcat(filePath, ...
83              'Dreadnaut_Output/Original/Cycle/aa',fileLabel,'.out');
84          PermLayer = strcat(filePath, ...
85             'Dreadnaut_Output/Reduced/Permutation/aa',fileLabel,'.out');
86          CycleLayer = strcat(filePath, ...
87              'Dreadnaut_Output/Reduced/Cycle/aa',fileLabel,'.out');
88          junk = strcat(filePath,'Junk/junk',fileLabel);
89          gapInput = ...
                strcat(filePath,'Gap_Input/gap',fileLabel,'.txt');
90          tic;
91          A = Constraintoa(k,s,t)';
92          fprintf(ftime,strcat(num2str(k),',',num2str(s),',', ...
93              num2str(t),',',num2str(toc),','));
94
95          %----Projection matrix using Moore Penrose ...
                pseudoinverse----
96          tic;
97          P = A * pinv(A'*A) * A';
98          fprintf(ftime,strcat(num2str(toc),','));
99          clear A;
100
101         %----Save lower half of P as a vector in a text file-----
102         tic;
103         p = zeros(1,sum(1:size(P,1)));
104         for n = 1:1:size(P,1)
105             p(1,(sum(1:(n-1))+1):(sum(1:(n-1))+n)) = P(n,1:n);
106         end;
107         clear P;
108         fp = fopen(PfileName,'w');
```

```
109             fprintf(fp,'%.11f ',p);
110             fclose(fp);
111             fprintf(ftime,strcat(num2str(toc),','));
112
113             clear p m n j;
114
115             %----Call Hnosigneff----
116             tic;
117             lin_com = ...
                    strcat(Hnosigneff,32,PfileName,32,g6fileName,32, ...
118                  num2str(s^k));
119             fprintf(fcom,strcat(lin_com,'\n'));
120
121             [status, result] = system(lin_com);
122             if status ~= 0
123                 disp(strcat('Error from Hnosigneff:',10,result,10));
124                 fprintf(ftime,'\n');
125                 fprintf(fsizes,'\n');
126                 break;
127             end;
128             fprintf(ftime,strcat(num2str(toc),','));
129
130             %---Load value for dfix----
131             j = 1;
132             while j < (length(result)-8) && ...
133                     ~strcmp(result(j:(j + 6)),'ncolor=')
134                 j = j + 1;
135             end;
136             j = j + 7;
137             n = j;
138             while n < length(result) && ~strcmp(result(n),':')
139                 n = n + 1;
```

```
140             end;
141             dfix = str2double(strrep(result(j:n),':',''));
142
143             %----Call listg----
144             tic;
145             lin_com = strcat(listg,32,g6fileName);
146             fprintf(fcom,strcat(lin_com,'\n'));
147
148             [status, result] = system(lin_com);
149             if status ~= 0
150                 disp(strcat('Error from listg:',10,result,10));
151                 fprintf(ftime,'\n');
152                 fprintf(fsizes,'\n');
153                 break;
154             end;
155             fprintf(ftime,strcat(num2str(toc),','));
156             fjunk = fopen(junk,'w');
157             fprintf(fjunk,result);
158             fclose(fjunk);
159
160             %---Make input file for dreadnaut (permutations)----
161             fperm = fopen(dreadInPerm, 'w');
162             fprintf(fperm,strcat('<',32,junk,'\np\n>',32,dreadOutPerm, ...
                    ...
163                  '\n?\nx\n'));
164             fclose(fperm);
165
166             %---Call dreadnaut (permutations)----
167 %            tic;
168 %            lin_com = strcat(dreadnaut,32,dreadInPerm);
169 %            fprintf(fcom,strcat(lin_com,'\n'));
170 %            [~, ~] = system(lin_com);
```

```
171 %             toc;
172
173             %---Make input file for dreadnaut (cycles)----
174             fcyc = fopen(dreadInCycle, 'w');
175             fprintf(fcyc,strcat('<',32,junk,'\n?\nx\n'));
176             fclose(fcyc);
177
178             %---Call dreadnaut (cycles)----
179             tic;
180             lin_com = strcat(dreadnaut,32,dreadInCycle);
181             fprintf(fcom,strcat(lin_com,'\n'));
182
183             [status, result] = system(lin_com);
184             if status ~= 0
185                 disp(strcat('Error from dreadnaut:',10,result,10));
186                 fprintf(ftime,'\n');
187                 fprintf(fsizes,'\n');
188                 break;
189             end;
190             fread = fopen(dreadOutCycle,'w');
191             fprintf(fread,result);
192             fclose(fread);
193             fprintf(ftime,strcat(num2str(toc),','));
194
195             %---Remove layers (permutations)---
196 %             tic;
197 %             RemoveLayers(dfix, dreadOutPerm, PermLayer, 0);
198 %             toc;
199
200             %---Remove layers (cycles)---
201
202             tic;
```

```
203             RemoveLayers(dfix, dreadOutCycle, CycleLayer, 1);
204             fprintf(ftime,strcat(num2str(toc),','));
205
206             %---Create Gap Input file---
207             result = num2str(StabilizerGroup(k, s, t));
208             while length(strrep(result,'  ','')) < length(result)
209                 result = strrep(result,'  ',' ');
210             end;
211             result = strrep(strtrim(result),' ',',');
212             fgap = fopen(gapInput, 'w');
213             fprintf(fgap,strcat('Read("',CycleLayer, ...
214                 '");\nSize(sys);\nSize(Stabilizer(sys,[', result, ...
215                 '],OnSets));\nquit;\n'));
216             fclose(fgap);
217
218             %---Run Gap to get size---
219             tic;
220             lin_com = strcat(gap,32,'<',32,gapInput);
221             fprintf(fcom,strcat(lin_com,'\n'));
222
223             [status, result] = system(lin_com);
224             result = FormatResult(result);
225             if status ~= 0 || max(size(result)) ~= 2 %#ok<BDSCA>
226                 disp(strcat('Error from gap:',10,result,10));
227                 fprintf(ftime,'\n');
228                 fprintf(fsizes,'\n');
229                 break;
230             end;
231
232             fprintf(fsizes,strcat(num2str(k),',',num2str(s),',', ...
233                 num2str(t),',', num2str(dfix),',', ...
234                 num2str(factorial(k)*(factorial(s-1))^k),',', ...
```

```
235                 num2str(result(2)),',',num2str((k+1)*factorial(k)*...
236                 (factorial(s))^k),',',num2str(result(1)),'\n'));
237             fprintf(ftime,strcat(num2str(toc),'\n'));  % (k+1)k!(s!)^k
238         end;
239     end;
240 end;
241
242 fclose(fcom);
243 fclose(fsizes);
244 fclose(ftime);
```

Appendix B: Computational Research Computer Code

B.1 Code For Table 4.1 Column 3

Appendix2/BF6GLP.m

```
function BF6GLP(nautyFile, Hnosigneff, k, s, t )

AAout = '';          %#ok<NASGU> Set the output to an empty string ... (default)

numDigit = 12;                                % Number of digits for ... precision

%---UNIX command line passes all inputs as strings, convert to ... integers----
if ischar(k)
    k = str2double(k);
end;

if ischar(s)
    s = str2double(s);
end;

if ischar(t)
    t = str2double(t);
end;

%----Determine folder containing the m file
[~, filePath] = system('pwd');
addpath(filePath);

```

```
%----Set intermediate file names----
fileLabel = strcat(num2str(k),'.',num2str(s),'.', num2str(t));
aaSizefileName = strcat(filePath,'/aa', fileLabel,'.naught.size');
PfileName = strcat(filePath,'/P', fileLabel,'.txt');
g6fileName = strcat(filePath,'/P',fileLabel,'.g6');
dreadInPerm = strcat(filePath,'/aa', fileLabel,'.naut.inp');
dreadOutPerm = strcat(filePath,'/aa', fileLabel,'.out');
dreadOutLayer = strcat(filePath,'/aa', fileLabel,'.red.out');
junk = strcat(filePath,'/junk',fileLabel,'.txt');
aagrp = strcat(filePath,'/aa', fileLabel,'.grp');

%----Nauty Linux command line scripts----
listg = strcat(nautyFile,'listg -d');              % listg executable
dreadnaut = strcat(nautyFile,'dreadnaut <');       % dreadnaut executable

%----Generate Rosenberg constraint matrix----
%[~,A,~] = Rosenberg(k,s,t,1);
%A = [eye(size(A,1)), -1*A]';

A = Constraintoa(k,s,t)';

%----Projection matrix using Moore Penrose pseudoinverse----
P = A * pinv(A'*A) * A';
clear A;

%----Save lower half of P as a vector in a text file-----
p = zeros(1,sum(1:size(P,1)));
for n = 1:1:size(P,1)
    p(1,(sum(1:(n-1))+1):(sum(1:(n-1))+n)) = P(n,1:n);
end;
for n = 1:1:size(p,2)
    if abs(p(1,n)) <= 10^(-1*numDigit)
```

```
56          p(1,n) = 0;
57      end;
58  end;
59  clear P;
60
61  fp = fopen(PfileName,'w');
62  fprintf(fp,strcat('%.',num2str(numDigit),'f',32),p);
63  fclose(fp);
64
65  clear p n j;
66
67  %----Call Hnosigneff----
68  lin_com = strcat(Hnosigneff,32, PfileName,32, g6fileName,32, ...
        num2str(s^k));
69
70  [status, result] = system(lin_com);
71  if status ~= 0
72      AAout = strcat('Error from Hnosigneff:',10,result,10);
73      disp(AAout);
74      return;
75  end;
76
77   %---Load value for dfix----
78  j = 1;
79  while j < (length(result)-1) && ...
80          ~strcmp(result(j:(j + 1)),'d=')
81      j = j + 1;
82  end;
83  j = j + 2;
84  q = j;
85  while q < length(result) && ~strcmp(result(q),' ')
86      q = q + 1;
```

```
87 end;
88 dfix = str2double(strrep(result(j:q),' ',''));
89
90 %----Call listg----
91 lin_com = strcat(listg,32,g6fileName,32,'>',32,junk);
92
93 [status, ~] = system(lin_com);
94 if status ~= 0
95     AAout = strcat('Error from listg:',10,result,10);
96     disp(AAout);
97     return;
98 end;
99
100 %---Make input file for dreadnaut (permutations)----
101 fperm = fopen(dreadInPerm, 'w');
102 fprintf(fperm,strcat('<',32,junk,'\np\n?\nx\n'));
103 fclose(fperm);
104
105 %---Call dreadnaut (cycles)----
106 lin_com = strcat(dreadnaut,32,dreadInPerm);
107
108 [status, AAout] = system(lin_com);
109
110 %---Get rid of 'Mode=dense' in nauty output so it can be read by ...
       Margot---
111 AAout = strrep(AAout,'Mode=dense ','');
112
113 if status ~= 0
114     AAout = strcat('Error from dreadnaut:',10,AAout,10);
115     disp(AAout);
116     return;
117 end;
```

```
118
119 %---Make output file for dreadnaut (cycles)----
120 fout = fopen(dreadOutPerm, 'w');
121 fprintf(fout,AAout);
122 fclose(fout);
123
124 %---Remove layers (cycles)---
125 GLPRemoveLayers(dfix, s^k, dreadOutPerm, dreadOutLayer);
126
127 %---create aa.naught.size file---
128 dreadOutLayer = strrep(dreadOutLayer, strcat(filePath,'/'),'');
129 fsize = fopen(aaSizefileName, 'w');
130 fprintf(fsize,strcat(dreadOutLayer,'\n', num2str(s^k),'\n','aa', ...
       fileLabel,'.grp','\n'));
131 fclose(fsize);
132
133 %---Call Margot script----
134 [~, ~] = system(strcat('/scratch/Obj2/Margot/gen_group <', ...
       32,aaSizefileName));
135
136 %---Copy to bidon.grp---
137 [~, ~] = system(strcat('rm',32,'bidon.grp'));
138 [~, ~] = system(strcat('cp',32,aagrp,32,'bidon.grp'));
139
140 end
```

B.2 Code For Table 4.1 Column 4

Appendix2/BF7GLP.m

```
1 function BF7GLP(nautyFile, Hnosigneff, N, k, s, t )
2
```

```
3 AAout = '';          %#ok<NASGU> Set the output to an empty string ...
      (default)
4
5 %---UNIX command line passes all inputs as strings, convert to ...
      integers----
6 if ischar(N)
7     N = str2double(N);
8 end;
9
10 if ischar(k)
11     k = str2double(k);
12 end;
13
14 if ischar(s)
15     s = str2double(s);
16 end;
17
18 if ischar(t)
19     t = str2double(t);
20 end;
21
22 %----Determine folder containing the m file
23 [~, filePath] = system('pwd');
24 addpath(filePath);
25
26 %----Set intermediate file names----
27 fileLabel = strcat(num2str(N), '.',num2str(k), '.',num2str(s), '.', ...
      num2str(t));
28 aaSizefileName = strcat(filePath, '/aa', fileLabel, '.naught.size');
29 PfileName = strcat(filePath,'/P', fileLabel,'.txt');
30 g6fileName = strcat(filePath,'/P', fileLabel,'.g6');
31 dreadInPerm = strcat(filePath,'/aa', fileLabel,'.naut.inp');
```

```
dreadOutPerm = strcat(filePath,'/aa', fileLabel,'.out');
dreadOutLayer = strcat(filePath,'/aa', fileLabel,'.red.out');
junk = strcat(filePath,'/junk', fileLabel,'.txt');
aagrp = strcat(filePath,'/aa', fileLabel,'.grp');
myLP = strcat(filePath,'/OA', fileLabel,'.mylp');

%----Nauty Linux command line scripts----
listg = strcat(nautyFile,'listg -d');              % listg executable
dreadnaut = strcat(nautyFile,'dreadnaut <');       % dreadnaut executable

%----Generate Rosenberg constraint matrix----
[~,A,b] = Rosenberg(k,s,t,floor(N/s^t));
A = -1* A;
b = -1* b;
m = size(A,1);
n = size(A,2);
c = zeros(1,n);
eq = -1*ones(m,1);

%---Modify bidon.inp file---
bidoninp = strcat(filePath,'/bidon.inp');
newbidoninp = strcat(filePath,'/newbidon.inp');
fb = fopen(bidoninp,'r');
tline = fgets(fb);
if ischar(tline)
    fn = fopen(newbidoninp,'w');
    fprintf(fn,tline);
    tline = fgets(fb);
    fprintf(fn,strcat(tline,'\n',num2str(min([n,N+1])),'\n'));
    tline = fgets(fb); %#ok<NASGU>
    tline = fgets(fb);
    fprintf(fn,tline);
```

```
64      fclose(fb);
65      [~, ~] = system(strcat('mv',32,newbidoninp,32,bidoninp));
66  else
67      fclose(fb);
68  end;
69
70  %---Build the mylp file----
71  BuildMyILP( c, A, b, eq, 1, 0, myLP )
72
73  %----Matrix representing the graph----
74  %P = ILPAdjacencyMatrix( A, b, c );
75  P = [zeros(n) A'; A zeros(m)];
76  %A = [-1*A, eye(m)];
77  %P = A * pinv(A'*A) * A';
78  clear A b c eq;
79
80  %----Save lower half of P as a vector in a text file-----
81  fp = fopen(PfileName,'w');
82  p = num2str(P(1,1));
83  fprintf(fp,p);
84  for i = 2:1:(m+n)
85      p = num2str(P(i,1:i) ,strcat('%i',32));
86      while length(p) > length(strrep(p,'  ',' '))
87          p = strrep(p,'  ',' ');
88      end;
89      fprintf(fp,strcat(32,p));
90  end;
91
92  fclose(fp);
93  clear P p i;
94
95  %----Call Hnosigneff----
```

```
lin_com = strcat(Hnosigneff,32,PfileName,32,g6fileName,32, ...
    num2str(s^k));

[status, result] = system(lin_com);
if status ~= 0
    AAout = strcat('Error from Hnosigneff:',10,result,10);
    disp(AAout);
    return;
end;

 %---Load value for dfix----
j = 1;
while j < (length(result)-1) && ...
        ~strcmp(result(j:(j + 1)),'d=')
    j = j + 1;
end;
j = j + 2;
q = j;
while q < length(result) && ~strcmp(result(q),' ')
    q = q + 1;
end;
dfix = str2double(strrep(result(j:q),' ',''));

%----Call listg----
lin_com = strcat(listg,32,g6fileName,32,'>',32,junk);

[status, ~] = system(lin_com);
if status ~= 0
    AAout = strcat('Error from listg:',10,result,10);
    disp(AAout);
    return;
end;
```

```
128
129 %---Make input file for dreadnaut (permutations)----
130 fperm = fopen(dreadInPerm, 'w');
131 fprintf(fperm,strcat('<',32,junk,'\np\n?\nx\n'));
132 fclose(fperm);
133
134 %---Call dreadnaut (cycles)----
135 lin_com = strcat(dreadnaut,32,dreadInPerm);
136
137 [status, AAout] = system(lin_com);
138
139 %---Get rid of 'Mode=dense' in nauty output so it can be read by ...
       Margot---
140 AAout = strrep(AAout,'Mode=dense ','');
141
142 if status ~= 0
143     AAout = strcat('Error from dreadnaut:',10,AAout,10);
144     disp(AAout);
145     return;
146 end;
147
148 %---Make output file for dreadnaut (permutations)----
149 fout = fopen(dreadOutPerm, 'w');
150 fprintf(fout,AAout);
151 fclose(fout);
152
153 %---Remove layers (cycles)---
154 GLPRemoveLayers(dfix, s^k, dreadOutPerm, dreadOutLayer);
155
156 %---create aa.naught.size file---
157 dreadOutLayer = strrep(dreadOutLayer,strcat(filePath,'/'),'');
158 fsize = fopen(aaSizefileName, 'w');
```

```
159 fprintf(fsize,strcat(dreadOutLayer, '\n',num2str(n), '\n', ...
        'aa',fileLabel,'.grp','\n'));
160 fclose(fsize);
161
162 %---Call Margot script----
163 [~, ~] = system(strcat( '/scratch/Obj2/Margot/gen_group <', 32, ...
        aaSizefileName));
164
165 %---Copy to bidon.grp---
166 [~, ~] = system(strcat('rm',32,'bidon.grp'));
167 [~, ~] = system(strcat('cp',32,aagrp,32,'bidon.grp'));
168
169 %---Copy to bidon.mylp---
170 [~, ~] = system(strcat('rm',32,'bidon.mylp'));
171 [~, ~] = system(strcat('cp',32,myLP,32,'bidon.mylp'));
172
173 end
```

B.3 Code For Table 4.3

Appendix2/Obj1.m

```
1 clear; clc;
2
3 numDigit = 10;
4 QRnumDigit = 9;
5 mySMTP = 'ms-afit-03.afit.edu';
6 myEmail = 'andrew.geyer@afit.edu';
7 setpref('Internet','SMTP_Server',mySMTP);
8 setpref('Internet','E_mail',myEmail);
9
10 %----Determine folder containing the m file
11 filePath = mfilename('fullpath');
```

```
12 filePath = filePath(1:(length(filePath)-length(mfilename('')))); 
13 
14 %----Linux command line script for Hnosigneff executable----
15 Hnosigneff = strcat(filePath,'Hnosigneff2'); 
16 
17 %----Nauty Linux command line scripts----
18 %nautyFile = '/home/andrew/Documents/nauty25/';       % Folder ...
       containing nauty
19 nautyFile = '/usr/local/nauty25/';       % Folder containing nauty
20 listg = strcat(nautyFile,'listg -d');     % listg executable
21 dreadnaut = strcat(nautyFile,'dreadnaut <');      % dreadnaut executable
22 
23 [~,~] = system(strcat('mkdir',32, filePath,'results')); 
24 addpath(strcat(filePath,'results/')); 
25 Linux_Command = strcat(filePath,'results/Linux_Command.txt'); 
26 fcom = fopen(Linux_Command,'w'); 
27 
28 %----Make subdirectories for files----
29 [~,~] = system(strcat('mkdir',32, filePath, 'results/Graph_Matrices')); 
30 addpath(strcat(filePath,'results/Graph_Matrices')); 
31 [~,~] = system(strcat('mkdir',32, filePath, 'results/g6_Files')); 
32 addpath(strcat(filePath,'results/g6_Files')); 
33 [~,~] = system(strcat('mkdir',32, filePath, 'results/Junk')); 
34 addpath(strcat(filePath,'results/Junk')); 
35 [~,~] = system(strcat('mkdir',32, filePath, 'results/Dreadnaut_Input')); 
36 addpath(strcat(filePath,'results/Dreadnaut_Input')); 
37 [~,~] = system(strcat('mkdir',32, filePath, ...
       'results/Dreadnaut_Input/Cycle')); 
38 addpath(strcat(filePath,'results/Dreadnaut_Input/Cycle')); 
39 [~,~] = system(strcat('mkdir',32, filePath, ...
       'results/Dreadnaut_Output')); 
40 addpath(strcat(filePath,'results/Dreadnaut_Output')); 
```

```
41 [~,~] = system(strcat('mkdir',32, filePath, ...
       'results/Dreadnaut_Output/Cycle'));
42 addpath(strcat(filePath,'results/Dreadnaut_Output/Cycle'));
43
44 %----List of permutation group sizes----
45 fsizes = fopen(strcat(filePath,'Group_Sizes.csv'),'w');
46 fprintf(fsizes,strcat('N,k,s,t, M dfix,M Group Size, rows B, columns ...
       B,', 'summation, rank(reduced B), rank(full B),p diff\n'));
47
48 % bsol file names
49 bsol = [' bsol32.6.2.4';
50          ' bsol80.6.2.4';
51          ' bsol64.6.2.4';
52          'bsol112.6.2.4';
53          ' bsol64.7.2.4';
54          ' bsol24.7.2.3';
55          ' bsol96.7.2.4';
56          ' bsol64.8.2.4';
57          ' bsol24.8.2.3';
58          ' bsol32.7.2.3';
59          ' bsol24.9.2.3';
60          ' bsol32.8.2.3'];
61
62 numbsol = size(bsol,1);
63
64 for i = 1:1:numbsol
65     % Set to the current bsol file path na
66
67
68     inputFile = strcat(filePath,'bsol/',strtrim(bsol(i,:)));
69
70     % Find number of factors
```

```
71      k =str2num(strrep(strrep(strtrim(bsol(i,:)), 'bsol',''), ...
            '.',char(32))); %#ok<ST2NM>
72      N = k(1);
73      s = k(3);
74      t = k(4);
75      k = k(2);
76
77      fileLabel = strcat('N', num2str(N),'k', num2str(k),'s', ...
            num2str(s),'t', num2str(t));
78      PfileName = strcat(filePath, 'results/Graph_Matrices/P', ...
            fileLabel,'.txt');
79      g6fileName = strcat(filePath, 'results/g6_Files/P', ...
            fileLabel,'.g6');
80      dreadInCycle = strcat(filePath, ...
            'results/Dreadnaut_Input/Cycle/aa', fileLabel,'.naut.inp');
81      dreadOutCycle = strcat(filePath, ...
            'results/Dreadnaut_Output/Cycle/aa', fileLabel,'.out');
82      junk = strcat(filePath, 'results/Junk/junk', fileLabel);
83
84      % Read in M file
85      [P,rowB,colB,rankB] = inputMP(inputFile, k, s,t, QRnumDigit);
86
87      %----Save lower half of P as a vector in a text file-----
88      p = zeros(1,sum(1:size(P,1)));
89      for j = 1:1:size(P,1)
90          p(1,(sum(1:(j-1))+1):(sum(1:(j-1))+j)) = P(j,1:j);
91      end;
92      clear P;
93      fp = fopen(PfileName,'w');
94      fprintf(fp,strcat('%.', num2str(numDigit), 'f', 32),p);
95      fclose(fp);
96
```

```
97      p_entries = sort(unique(round(p .* (10^numDigit)) ./ ...
            (10^numDigit)));
98      p_diff = p_entries(1,2) - p_entries(1,1);
99      for j = 3:1:max(size(p_entries))
100         if p_entries(1,j) - p_entries(1,j-1) < p_diff
101             p_diff = p_entries(1,j) - p_entries(1,j-1);
102         end;
103     end;
104
105     clear p j p_entries;
106
107     %----Call Hnosigneff----
108     lin_com = strcat(Hnosigneff, 32, PfileName, 32, g6fileName, 32, ...
            num2str(rowB));
109     fprintf(fcom,strcat(lin_com,'\n'));
110
111     [status, result] = system(lin_com);
112     if status ~= 0
113         disp(strcat('Error from Hnosigneff2:', 10, result, 10));
114         sendmail(myEmail,fileLabel, 'Error from Hnosigneff2.')
115         fprintf(fsizes,'\n');
116         break;
117     end;
118
119     %---Load value for dfix----
120     j = 1;
121     while j < (length(result)-1) && ...
122             ~strcmp(result(j:(j + 1)),'d=')
123         j = j + 1;
124     end;
125     j = j + 2;
126     q = j;
```

```
127     while q < length(result) && ~strcmp(result(q),char(32))
128         q = q + 1;
129     end;
130     dfix = str2double(strrep(result(j:q), ':',''));
131
132     %----Call listg----
133     lin_com = strcat(listg, 32, g6fileName, 32, '>' ,32, junk);
134     fprintf(fcom,strcat(lin_com,'\n'));
135
136     [status, ~] = system(lin_com);
137     if status ~= 0
138         disp(strcat('Error from listg:', 10, result, 10));
139         sendmail(myEmail,fileLabel, 'Error from listg.')
140         fprintf(fsizes,'\n');
141         break;
142     end;
143
144     %---Make input file for dreadnaut (cycles)----
145     fcyc = fopen(dreadInCycle, 'w');
146     fprintf(fcyc,strcat('<', 32, junk, '\n?\nx\n'));
147     fclose(fcyc);
148
149     %---Call dreadnaut (cycles)----
150     lin_com = strcat(dreadnaut, 32, dreadInCycle, 32, '>', 32, ...
            dreadOutCycle);
151     fprintf(fcom,strcat(lin_com,'\n'));
152     [status, result] = system(lin_com);
153
154     if status ~= 0
155         disp(strcat('Error from dreadnaut:', 10, result, 10));
156         sendmail(myEmail,fileLabel, 'Error from dreadnaut.')
157         fprintf(fsizes,'\n');
```

```
158         break;
159     end;
160     fread = fopen(dreadOutCycle);
161     result = fgets(fread);
162     while ischar(result)
163         if length(strrep(result, 'grpsize=','')) < length(result)
164             break;
165         end;
166         result = fgets(fread);
167     end;
168     fclose(fread);
169
170     %---find the group size in the dreadnaut result---
171     j = length(result)-8;
172     while j > 1 && ~strcmp(result(j:(j+7)), 'grpsize=')
173         j = j - 1;
174     end;
175     if j < 1
176         disp('Unable to find group size in dreadnaut output.');
177         sendmail(myEmail,fileLabel, 'Unable to find group size in ...
                dreadnaut output.');
178         break;
179     end;
180     j = j + 8;
181     q = j;
182     while q < length(result)-1 && ~strcmp(';',result(q+1))
183         q = q + 1;
184     end;
185
186     ksum = 0;
187     for z=t+1:1:k
188         ksum = ksum+(s-1)^z * nchoosek(k,z);
```

```
189     end;
190
191     B = inputM(inputFile,N,k,s);
192
193     fprintf(fsizes, strcat(num2str(N), ',', num2str(k), ',', ...
            num2str(s), ',', num2str(t), ',', num2str(dfix),',', ...
            num2str(result(j:q)),',', num2str(rowB),',', num2str(colB), ...
            ',', num2str(ksum), ',', num2str(rankB), ',', ...
            num2str(rank(B)), ',', num2str(p_diff),'\n'));
194
195     sendmail(myEmail, fileLabel, 'Another case is done.', ...
            strcat(filePath, 'Group_Sizes.csv'));
196
197 end;
```

Appendix2/inputM.m

```
1  function [ M ] = inputM( fileName,N,k,s)
2  % Reads the bsol file to get the M matrix
3  % fileName is the path to the M matrix file
4  % k is the number of factors
5
6  fin = fopen(fileName);
7  tline = fgetl(fin);
8  Mline = [];
9  M = [];
10
11 while ischar(tline)
12      Mline = [Mline str2num(tline)]; %#ok<ST2NM>
13      Msize = max(size(Mline));
14      if Msize >= s^k
15          newMline = [];
16          if Msize > s^k
```

```
17              newMline = Mline(1,s^k+1:Msize);
18              Mline = Mline(1,1:s^k);
19          end;
20          M = [M (Mline - N/s^k)'];  % N = sum(Mline)/s^t
21          Mline = newMline;
22      end;
23      tline = fgetl(fin);
24  end;
25
26  fclose(fin);
27
28  end
```

Appendix2/inputMP.m

```
1  function [ P, m, n, rankB ] = inputMP( fileName ,k,s,t, numDigit)
2  % Reads the bsol file to get the M matrix
3  % fileName is the path to the M matrix file
4  % k is the number of factors
5
6  vecStep = 100;
7  fin = fopen(fileName);
8  tline = fgetl(fin);
9  Mline = [];
10 B = [];
11 numVec = 0;
12
13 while ischar(tline)
14      Mline = [Mline str2num(tline)]; %#ok<ST2NM>
15      Msize = max(size(Mline));
16      if Msize >= s^k
17          newMline = [];
18          if Msize > s^k
```

```
19              newMline = Mline(1,s^k+1:Msize);
20              Mline = Mline(1,1:s^k);
21          end;
22          B = [B (Mline - sum(Mline)/s^k)'];  % N = sum(Mline)/s^t
23          Mline = newMline;
24          numVec = numVec + 1;
25      end;
26      tline = fgetl(fin);
27      if numVec == vecStep || ~ischar(tline)
28          i = 1;
29          [q,r] = qr(B);
30          rInd = [];
31          for j = 1:size(r,2)
32              if(abs(r(i,j)) >10^(-1*numDigit))
33                  rInd = [rInd r(:,j)];
34                  i = i + 1;
35              end;
36              if(i > size(r,1))
37                  break;
38              end;
39          end;
40          clear r;
41          B=q*rInd;  %here's your answer
42          clear q rInd;
43          numVec = 0;
44      end;
45  end;
46
47  fclose(fin);
48
49  [m n] = size(B);
50
```

```
51 rankB = rank(B);
52
53 P = B*pinv(B'*B)*B';
54
55 clear B;
56
57 end
```

B.4 Code to Compute $H_{s,k}^{LP}$

Appendix2/HYBRID5for6GLP.m

```
1 function HYBRID5for6GLP(nautyFile, Hnosigneff, Afile, ClassConst, k, ...
      s, t )
2
3 AAout = '';          %#ok<NASGU> Set the output to an empty string ...
      (default)
4
5 numDigit = 12;                              % Number of digits for ...
      precision
6
7 %---UNIX command line passes all inputs as strings, convert to ...
      integers----
8 if ischar(k)
9     k = str2double(k);
10 end;
11
12 if ischar(s)
13     s = str2double(s);
14 end;
15
16 if ischar(t)
17     t = str2double(t);
```

```
18 end;
19
20 %----Determine folder containing the m file
21 [~, filePath] = system('pwd');
22 addpath(filePath);
23
24 %----Set intermediate file names----
25 fileLabel = strcat(num2str(k-1),'.',num2str(s),'.', num2str(t));
26 aaSizefileName = strcat(filePath,'/aa', fileLabel, '.naught.size');
27 PfileName = strcat(filePath,'/P', fileLabel,'.txt');
28 g6fileName = strcat(filePath,'/P',fileLabel,'.g6');
29 dreadInPerm = strcat(filePath,'/aa', fileLabel, '.naut.inp');
30 dreadOutPerm = strcat(filePath,'/aa', fileLabel,'.out');
31 dreadOutLayer = strcat(filePath,'/aa', fileLabel,'.red.out');
32 junk = strcat(filePath,'/junk', fileLabel,'.txt');
33 aagrp = strcat(filePath,'/aa', fileLabel,'.grp');
34
35 %----Nauty Linux command line scripts----
36 listg = strcat(nautyFile,'listg -d');              % listg executable
37 dreadnaut = strcat(nautyFile,'dreadnaut <');      % dreadnaut executable
38
39 %----Read in the constraint matrix----
40 A = [];
41 fAA = fopen(strcat(filePath,'/',Afile), 'r');
42 tline = fgets(fAA);
43 while ischar(tline)
44     a = zeros(1,s^(k-1));
45     ind = str2num(tline); %#ok<ST2NM>
46     ind = ind(2:size(ind,2))+1;
47     a(ind) = 1;
48     A = [A;a]; %#ok<AGROW>
49     tline = fgets(fAA);
```

```
50 end;
51
52 %dlmwrite(strcat(pwd,'/Atest.txt'),A);
53
54 %----Projection matrix using Moore Penrose pseudoinverse----
55 P = A' * pinv(A*A') * A;
56 clear A;
57
58 %----Save lower half of P as a vector in a text file-----
59 p = zeros(1,sum(1:size(P,1)));
60 for n = 1:1:size(P,1)
61     p(1,(sum(1:(n-1))+1):(sum(1:(n-1))+n)) = P(n,1:n);
62 end;
63 for n = 1:1:size(p,2)
64     if abs(p(1,n)) <= 10^(-1*numDigit)
65         p(1,n) = 0;
66     end;
67 end;
68 clear P;
69
70 fp = fopen(PfileName,'w');
71 fprintf(fp,strcat('%.',num2str(numDigit),'f',32),p);
72 fclose(fp);
73
74 clear p n j;
75
76 %----Call Hnosigneff----
77 lin_com = strcat(Hnosigneff, 32, PfileName, 32, g6fileName, 32, ...
      num2str(s^(k-1)), 32, num2str(10^-6));
78
79 [status, result] = system(lin_com);
80 if status ~= 0
```

```
81      AAout = strcat('Error from Hnosigneff:', 10, result, 10);
82      disp(AAout);
83      return;
84  end;
85
86   %---Load value for dfix----
87  j = 1;
88  while j < (length(result)-1) && ~strcmp(result(j:(j + 1)),'d=')
89      j = j + 1;
90  end;
91  j = j + 2;
92  q = j;
93  while q < length(result) && ~strcmp(result(q),' ')
94      q = q + 1;
95  end;
96  dfix = str2double(strrep(result(j:q), ' ', ''));
97
98  %----Call listg----
99  lin_com = strcat(listg,32, g6fileName, 32, '>', 32, junk);
100
101 [status, ~] = system(lin_com);
102 if status ~= 0
103     AAout = strcat('Error from listg:', 10, result, 10);
104     disp(AAout);
105     return;
106 end;
107
108 %---Add color to the graph using the ClassConst file---
109 fcolor = 'f=[';
110 fclass = fopen(strcat(filePath, '/', ClassConst), 'r');
111 tline = fgets(fclass);
112 while ischar(tline)
```

```
113     a = str2num(tline); %#ok<ST2NM>
114     if max(size(a)) > 1
115         asize = max(size(a));
116         for i=2:1:asize
117             for j = (a(i) * dfix):1:((a(i) + 1) * dfix - 1)
118                 fcolor = strcat(fcolor, num2str(j), ',');
119             end;
120         end;
121         fcolor = strcat(fcolor(1:length(fcolor) - 1), '|');
122     end;
123     tline = fgets(fclass);
124 end;
125
126 fclose(fclass);
127
128 fcolor = strrep(strcat(fcolor(1:length(fcolor) - 1),']'), '|', ' | ');
129
130 %---Make input file for dreadnaut (permutations)----
131 fperm = fopen(dreadInPerm, 'w');
132 fprintf(fperm, strcat('<', 32, junk, '\n', fcolor, '\np\n?\nx\n'));
133 fclose(fperm);
134
135 %---Call dreadnaut (permutations)----
136 lin_com = strcat(dreadnaut, 32, dreadInPerm);
137
138 [status, AAout] = system(lin_com);
139
140 %---Get rid of 'Mode=dense' in nauty output so it can be read by ...
        Margot---
141 AAout = strrep(AAout, 'Mode=dense ', '');
142
143 if status ~= 0
```

```
144      AAout = strcat('Error from dreadnaut:', 10, AAout, 10);
145      disp(AAout);
146      return;
147  end;
148
149  %---Make output file for dreadnaut (permutations)----
150  fout = fopen(dreadOutPerm, 'w');
151  fprintf(fout,AAout);
152  fclose(fout);
153
154  %---Remove layers (cycles)---
155  GLPRemoveLayers(dfix, s^(k-1), dreadOutPerm, dreadOutLayer);
156
157  %---create aa.naught.size file---
158
159  faaout = fopen(strcat(filePath, '/new.aa.naught.size'), 'r');
160  tline = fgets(faaout);
161  tline = fgets(faaout);
162  dreadOutLayer = strrep(dreadOutLayer, strcat(filePath,'/'), '');
163  fsize = fopen(aaSizefileName, 'w');
164  fprintf(fsize,strcat(dreadOutLayer, '\n', tline, '\n', 'aa', ...
        fileLabel, '.grp', '\n'));
165  fclose(fsize);
166
167  %---Call Margot script----
168  [~, ~] = system(strcat('/scratch/Obj2/Margot/gen_group <', 32, ...
        aaSizefileName));
169
170  %---Copy to bidon.grp---
171  [~, ~] = system(strcat('rm',32, 'bidon.grp'));
172  [~, ~] = system(strcat('cp',32, aagrp, 32, 'bidon.grp'));
173
```

```
174 end
```

B.5 Code to Compute $H_{all_1}^{LP}$

Appendix2/HYBRID5for6GLPconst.m

```
 1 function HYBRID5for6GLPconst(nautyFile, Hnosigneff, Afile, ...
       ClassConst, k, s, t )
 2
 3 %---This function calculates the GLP group directly from the ...
       constraints in the bidon.mylp file.
 4
 5 AAout = '';          %#ok<NASGU> Set the output to an empty string ...
       (default)
 6
 7 numDigit = 12;                                % Number of digits for ...
       precision
 8
 9 %---UNIX command line passes all inputs as strings, convert to ...
       integers----
10 if ischar(k)
11     k = str2double(k);
12 end;
13
14 if ischar(s)
15     s = str2double(s);
16 end;
17
18 if ischar(t)
19     t = str2double(t);
20 end;
21
22 %----Determine folder containing the m file
```

```
23 [~, filePath] = system('pwd');
24 addpath(filePath);
25
26 %----Set intermediate file names----
27 fileLabel = strcat(num2str(k - 1), '.', num2str(s), '.', num2str(t));
28 aaSizefileName = strcat(filePath, '/aa', fileLabel, '.naught.size');
29 PfileName = strcat(filePath,'/P', fileLabel, '.txt');
30 g6fileName = strcat(filePath,'/P', fileLabel, '.g6');
31 dreadInPerm = strcat(filePath,'/aa', fileLabel, '.naut.inp');
32 dreadOutPerm = strcat(filePath,'/aa', fileLabel, '.out');
33 dreadOutLayer = strcat(filePath,'/aa', fileLabel, '.red.out');
34 junk = strcat(filePath,'/junk', fileLabel, '.txt');
35 aagrp = strcat(filePath,'/aa', fileLabel, '.grp');
36
37 %----Nauty Linux command line scripts----
38 listg = strcat(nautyFile, 'listg -d');              % listg executable
39 dreadnaut = strcat(nautyFile, 'dreadnaut <');       % dreadnaut ...
       executable
40
41 %----Read in the constraint matrix----
42 A = [];
43 fAA = fopen(strcat(filePath, '/', Afile), 'r');
44 tline = fgets(fAA); %#ok<NASGU>
45 count = str2num(fgets(fAA)); %#ok<ST2NM>
46 nvars = count(1);
47 tline = fgets(fAA); %#ok<NASGU>
48 tline = fgets(fAA); %#ok<NASGU>
49 tline = fgets(fAA);
50
51 while ischar(tline)
52     a = zeros(1, nvars);
53     ind = str2num(tline); %#ok<ST2NM>
```

```
54      ind = ind(2:size(ind, 2)) + 1;
55      a(ind) = 1;
56      A = [A;a]; %#ok<AGROW>
57      tline = fgets(fAA); %#ok<NASGU>
58      tline = fgets(fAA);
59  end;
60
61  % dlmwrite('Atest.txt', A);
62
63  %----Projection matrix using Moore Penrose pseudoinverse----
64  P = A' * pinv(A * A') * A;
65  clear A;
66
67  %----Save lower half of P as a vector in a text file-----
68  p = zeros(1,sum(1:size(P, 1)));
69  for n = 1:1:size(P,1)
70      p(1, (sum(1:(n - 1)) + 1):(sum(1:(n - 1)) + n)) = P(n, 1:n);
71  end;
72  for n = 1:1:size(p,2)
73      if abs(p(1, n)) <= 10^(-1 * numDigit)
74          p(1,n) = 0;
75      end;
76  end;
77  clear P;
78
79  fp = fopen(PfileName, 'w');
80  fprintf(fp,strcat('%.', num2str(numDigit), 'f', 32), p);
81  fclose(fp);
82
83  clear p n j;
84
85  %----Call Hnosigneff----
```

```
lin_com = strcat(Hnosigneff, 32, PfileName, 32, g6fileName, 32, ...
    num2str(nvars), 32, num2str(10^-6));

[status, result] = system(lin_com);
if status ~= 0
    AAout = strcat('Error from Hnosigneff:', 10, result,10);
    disp(AAout);
    return;
end;

 %---Load value for dfix----
j = 1;
while j < (length(result) - 1) && ~strcmp(result(j:(j + 1)), 'd=')
    j = j + 1;
end;
j = j + 2;
q = j;
while q < length(result) && ~strcmp(result(q), ' ')
    q = q + 1;
end;
dfix = str2double(strrep(result(j:q), ' ', ''));

%----Call listg----
lin_com = strcat(listg, 32, g6fileName, 32, '>', 32, junk);

[status, ~] = system(lin_com);
if status ~= 0
    AAout = strcat('Error from listg:', 10, result, 10);
    disp(AAout);
    return;
end;

```

```
117 %---Add color to the graph using the ClassConst file---
118 fcolor = 'f=[';
119 fclass = fopen(strcat(filePath, '/', ClassConst), 'r');
120 tline = fgets(fclass);
121 while ischar(tline)
122     a = str2num(tline); %#ok<ST2NM>
123     if max(size(a)) > 1
124         asize = max(size(a));
125         for i=2:1:asize
126             for j = (a(i) * dfix):1:((a(i) + 1) * dfix - 1)
127                 fcolor = strcat(fcolor, num2str(j), ',');
128             end;
129         end;
130         fcolor = strcat(fcolor(1:length(fcolor) - 1), '|');
131     end;
132     tline = fgets(fclass);
133 end;
134
135 fclose(fclass);
136
137 fcolor = strrep(strcat(fcolor(1:length(fcolor) - 1), ']'), '|', ' | ');
138
139 %---Make input file for dreadnaut (permutations)----
140 fperm = fopen(dreadInPerm, 'w');
141 fprintf(fperm,strcat('<', 32, junk, '\n', fcolor, '\np\n?\nx\n'));
142 fclose(fperm);
143
144 %---Call dreadnaut (cycles)----
145 lin_com = strcat(dreadnaut, 32, dreadInPerm);
146
147 [status, AAout] = system(lin_com);
148
```

```
149 %---Get rid of 'Mode=dense' in nauty output so it can be read by ...
      Margot---
150 AAout = strrep(AAout, 'Mode=dense ', '');
151
152 if status ~= 0
153     AAout = strcat('Error from dreadnaut:', 10, AAout, 10);
154     disp(AAout);
155     return;
156 end;
157
158 %---Make output file for dreadnaut (cycles)----
159 fout = fopen(dreadOutPerm, 'w');
160 fprintf(fout,AAout);
161 fclose(fout);
162
163 %---Remove layers (cycles)---
164 GLPRemoveLayers(dfix, nvars, dreadOutPerm, dreadOutLayer);
165
166 %---create aa.naught.size file---
167
168 faaout = fopen('new.aa.naught.size','r+');
169 tline = fgets(faaout); %#ok<NASGU>
170 tline = fgets(faaout);
171 dreadOutLayer = strrep(dreadOutLayer, strcat(filePath,'/'), '');
172 fsize = fopen(aaSizefileName, 'w');
173 fprintf(fsize,strcat(dreadOutLayer, '\n', tline, '\n', 'aa', ...
      fileLabel, '.grp','\n'));
174 fclose(fsize);
175
176 %---Call Margot script----
177 [~, ~] = system(strcat('/scratch/Obj2/Margot/gen_group <', 32, ...
      aaSizefileName));
```

```
178
179 %---Copy to bidon.grp---
180 [~, ~] = system(strcat('rm', 32, 'bidon.grp'));
181 [~, ~] = system(strcat('cp', 32, aagrp, 32, 'bidon.grp'));
182
183 end
```

B.6 Code For Table 4.2

Appendix2/DoubleCosetTest.m

```
1 clear; clc;
2
3 ProbList = {'Coak6s2t2lam5.lp.gz';
4             'Rosk6s2t2lam5.lp.gz';
5             'Coak7s2t2lam5.lp.gz';
6             'Rosk7s2t2lam5.lp.gz';
7             'Coak8s2t2lam5.lp.gz';
8             'Rosk8s2t2lam5.lp.gz';
9             'Coak9s2t2lam5.lp.gz';
10            'Rosk9s2t2lam5.lp.gz';
11            'Coak10s2t2lam5.lp.gz';
12            'Rosk10s2t2lam5.lp.gz';
13            'Coak5s2t2lam6.lp.gz';
14            'Rosk5s2t2lam6.lp.gz';
15            'Coak6s2t2lam6.lp.gz';
16            'Rosk6s2t2lam6.lp.gz';
17            'Coak7s2t2lam6.lp.gz';
18            'Rosk7s2t2lam6.lp.gz';
19            'Coak8s2t2lam6.lp.gz';
20            'Rosk8s2t2lam6.lp.gz';
21            'Coak9s2t2lam6.lp.gz';
22            'Rosk9s2t2lam6.lp.gz';
```

```
23              'Coak10s2t2lam6.lp.gz';
24              'Rosk10s2t2lam6.lp.gz';
25              'Coak11s2t2lam6.lp.gz';
26              'Rosk11s2t2lam6.lp.gz';
27              'Coak5s2t3lam3.lp.gz';
28              'Rosk5s2t3lam3.lp.gz';
29              'Coak6s2t3lam3.lp.gz';
30              'Rosk6s2t3lam3.lp.gz';
31              'Coak7s2t3lam3.lp.gz';
32              'Rosk7s2t3lam3.lp.gz';
33              'Coak8s2t3lam3.lp.gz';
34              'Rosk8s2t3lam3.lp.gz';
35              'Coak9s2t3lam3.lp.gz';
36              'Rosk9s2t3lam3.lp.gz';
37              'Coak10s2t3lam3.lp.gz';
38              'Rosk10s2t3lam3.lp.gz';
39              'Coak11s2t3lam3.lp.gz';
40              'Rosk11s2t3lam3.lp.gz';
41              'Coak6s2t3lam4.lp.gz';
42              'Rosk6s2t3lam4.lp.gz';
43              'Coak7s2t3lam4.lp.gz';
44              'Rosk7s2t3lam4.lp.gz';
45              'Coak8s2t3lam4.lp.gz';
46              'Rosk8s2t3lam4.lp.gz';
47              'Coak9s2t3lam4.lp.gz';
48              'Rosk9s2t3lam4.lp.gz';
49              'Coak10s2t3lam4.lp.gz';
50              'Rosk10s2t3lam4.lp.gz';
51              'Coak11s2t3lam4.lp.gz';
52              'Rosk11s2t3lam4.lp.gz';
53              'Coak6s2t3lam5.lp.gz';
54              'Rosk6s2t3lam5.lp.gz';
```

```
55              'Coak7s2t3lam5.lp.gz';
56              'Rosk7s2t3lam5.lp.gz';
57              'Coak8s2t3lam5.lp.gz';
58              'Rosk8s2t3lam5.lp.gz';
59              'Coak9s2t3lam5.lp.gz';
60              'Rosk9s2t3lam5.lp.gz';
61              'Coak10s2t3lam5.lp.gz';
62              'Rosk10s2t3lam5.lp.gz';
63              'Coak6s2t3lam6.lp.gz';
64              'Rosk6s2t3lam6.lp.gz';
65              'Coak7s2t3lam6.lp.gz';
66              'Rosk7s2t3lam6.lp.gz';
67              'Coak8s2t3lam6.lp.gz';
68              'Rosk8s2t3lam6.lp.gz';
69              'Coak5s3t3lam2.lp.gz';
70              'Rosk5s3t3lam2.lp.gz';
71              'Coak6s3t3lam2.lp.gz';
72              'Rosk6s3t3lam2.lp.gz';
73              'Coak6s2t3lam7.lp.gz';
74              'Rosk6s2t3lam7.lp.gz';
75              'Coak7s2t3lam7.lp.gz';
76              'Rosk7s2t3lam7.lp.gz';
77              'Coak7s2t4lam4.lp.gz';
78              'Rosk7s2t4lam4.lp.gz';
79              'Coak8s2t4lam4.lp.gz';
80              'Rosk8s2t4lam4.lp.gz';
81              'Coak6s2t4lam5.lp.gz';
82              'Rosk6s2t4lam5.lp.gz';
83              'Coak7s2t4lam5.lp.gz';
84              'Rosk7s2t4lam5.lp.gz';
85              'Coak5s3t4lam1.lp.gz';
86              'Rosk5s3t4lam1.lp.gz';
```

```
87              'Coak7s2t4lam6.lp.gz';
88              'Rosk7s2t4lam6.lp.gz';
89              'Coak8s2t4lam6.lp.gz';
90              'Rosk8s2t4lam6.lp.gz';
91              'Coak6s2t4lam7.lp.gz';
92              'Rosk6s2t4lam7.lp.gz';
93              'Coak7s2t4lam7.lp.gz';
94              'Rosk7s2t4lam7.lp.gz';
95              'Coak8s2t4lam9.lp.gz';
96              'Rosk8s2t4lam9.lp.gz';
97              'Coak9s2t4lam9.lp.gz';
98              'Rosk9s2t4lam9.lp.gz';
99              'Coak6s3t4lam2.lp.gz';
100             'Rosk6s3t4lam2.lp.gz';
101             'seymour.gz';
102             'mzzv42z.gz';
103             'air03.gz';
104             'arki001.gz';
105             'blend2.gz';
106             'enigma.gz';
107             'fiber.gz';
108             'gen.gz';
109             'glass4.gz';
110             'mas74.gz';
111             'mas76.gz';
112             'misc03.gz';
113             'misc06.gz';
114             'misc07.gz';
115             'mitre.gz';
116             'mkc.gz';
117             'mzzv11.gz';
118             'noswot.gz';
```

```
119             'opt1217.gz';
120             'p0201.gz';
121             'p2756.gz';
122             'protfold.gz';
123             'qiu.gz';
124             'rgn.gz';
125             'rout.gz';
126             'stein27.gz';
127             'swath.gz';
128             'timtab1.gz';
129             'timtab2.gz'};
130
131 numProb = size(ProbList,1);
132 addSlack = 1;
133
134 [~,hostname] = system('hostname');
135
136 if strcmp(cellstr(hostname),'Deep-Thought')
137     gap = '/etc/gap4r6/bin/gap-default64.sh -m 20000m -q';
138     matlabpool('open',3);
139 elseif strcmp(cellstr(hostname),'ensphd01')
140     gap = '/usr/local/gap4r5/gap.shi -m 20000m -q';
141     matlabpool('open',12);
142 else
143     gap = 'gap.sh -m 20000m -q';
144     matlabpool('open',12);
145 end;
146
147 parfor i = 1:numProb
148     ProbName = strrep(strrep(ProbList{i},'.gz',''),'.lp','');
149     filePath = strcat(ProbName,'/DoubleCoset/');
150     [~,~] = system(strcat('mkdir',32,filePath));
```

```
151     subNautyCycle = strcat(filePath,ProbName,'.aa.sub');
152     superNautyCycle = strcat(filePath,ProbName,'.aa.sup');
153     fileName = strcat(filePath,ProbName,'.csv');
154     fout = fopen(fileName,'w');
155     fileName = strcat(ProbName,'/',ProbList{i});
156
157     %-----------Create supergroup--------------------
158
159     fprintf(1,strcat(10,10,'Starting work on',32,ProbName,32,'GLP ...
            with cycles.',10,10));
160
161     [ cplex, numslack ] = ...
            BuildProblem(fileName,addSlack,[],[],[],[],[],[],[],'');
162     n = size(cplex.Model.A,2);
163     tic;
164     useAdj = 0;
165     [ P, numDigit ] = ProjMatrix( cplex.Model.A, cplex.Model.obj, ...
            useAdj );
166     if ~isempty(P)
167         [ superdfix,g6fileName, numP ] = runHnosigneff(P,numDigit, ...
168                                 filePath,ProbName);
169         if superdfix > 0
170             fcolor = ColorGraph(cplex.Model.obj,cplex.Model.lhs, ...
171                 cplex.Model.rhs,cplex.Model.lb,cplex.Model.ub, ...
172                 cplex.Model.ctype,superdfix,useAdj);
173             [ grpsize, ~ ] = NautyGroup( g6fileName,fcolor,1);
174             fprintf(fout,'%i,',[superdfix,numP]);
175             fprintf(fout,strcat(grpsize,','));
176             fprintf(fout,'%d,',toc);
177             fprintf(1,strcat(10,10,'Finishing work on',32,ProbName, ...
178                                                 32,'GLP with ...
                                                    cycles.',10,10));
```

```
179             [~,~] = system(strcat('mv',32,filePath, ...
180                                   ProbName, '.aa.out',32, ...
                                          superNautyCycle));
181
182             %----------------Make supergroup in Gap ...
                    format-----------------
183
184             G = Nauty2Gap(superdfix, n, superNautyCycle, 'G');
185
186             %-----------Create subgroup--------------------
187
188             fprintf(1,strcat(10,10, 'Starting work on',32, ...
                    ProbName,32, 'Margot with cycles.',10,10));
189
190             tic;
191             useAdj = 1;
192             [ P, numDigit ] = ProjMatrix( cplex.Model.A, ...
                    cplex.Model.obj, useAdj );
193             if ~isempty(P)
194                 [ subdfix,g6fileName, numP ] = ...
                        runHnosigneff(P,numDigit,filePath,ProbName);
195
196                 if subdfix > 0
197                     fcolor = ...
                            ColorGraph(cplex.Model.obj,cplex.Model.lhs, ...
198                             cplex.Model.rhs,cplex.Model.lb,cplex.Model.ub, ...
                                    ...
199                             cplex.Model.ctype,subdfix,useAdj);
200                     [ grpsize, ~ ] = NautyGroup( g6fileName,fcolor,1);
201                     fprintf(fout,'%i,',[subdfix,numP]);
202                     fprintf(fout,strcat(grpsize,','));
203                     fprintf(fout,'%d,',toc);
```

```
204                 [~,~] = system(strcat('mv',32, ...
                        filePath,ProbName, '.aa.out',32, ...
                        subNautyCycle));
205             else
206                 fprintf(fout,',,,,');
207             end;
208         end;
209         fprintf(1,strcat(10,10, 'Finishing work on', ...
                32,ProbName, 32,'Margot with cycles.',10,10));
210
211         %----------Supergroup orbit projection ...
                matrix------------------
212
213         E = OrbitProj( filePath,ProbName,n,G, 'G' );
214         x = [];
215         if ~isempty(E)
216             [~,~,exitflag] = cplexlp(cplex.Model.obj, [], [], ...
                    [cplex.Model.A;E], ...
217                     [cplex.Model.rhs;zeros(n,1)], ...
                            cplex.Model.lb,cplex.Model.ub);
218         end;
219         if exitflag == -2
220             fprintf(1,strcat(10,10,'Beginning', 32,ProbName,32, ...
                    'double coset decomposition.',10,10));
221             tic;
222             %------------Make subgroup in Gap ...
                    format-------------------
223             GAb = Nauty2Gap(subdfix, n, subNautyCycle, 'GAb');
224
225             E = OrbitProj( filePath,ProbName,n,GAb, 'GAb' );
226             x = [];
227             if ~isempty(E)
```

```
228                 [~,~,exitflag] = ...
                        cplexlp(cplex.Model.obj,[],[],[cplex.Model.A;E], ...
                        ...
229                         [cplex.Model.rhs;zeros(n,1)], ...
                                cplex.Model.lb,cplex.Model.ub);
230             end;
231             if exitflag ~= -2
232                 passcheck = 1;
233             else
234                 passcheck = 0;
235             end;
236             while passcheck == 1
237                 passcheck = 0;
238                 repFile = ...
                        DoubleCoset(filePath,ProbName,G,'G',GAb,'GAb');
239
240                 fin = fopen(repFile,'r');
241                 tline = fgets(fin);
242                 while ischar(tline)
243                 %---iterate through double coset ...
                        representatives-------
244                      if length(tline) - ...
                            length(strrep(strrep(tline, '(', '') ...
                            ,')', '')) >= 2
245                          Gtemp = ...
                                strrep(GAb,');',strcat(',',tline,');'));
246                          E = OrbitProj( ...
                                filePath,ProbName,n,Gtemp, 'GAb' );
247                          x = [];
248                          if ~isempty(E)
249                              [~,~,exitflag] = ...
                                    cplexlp(cplex.Model.obj,[],[], ...
```

```
250                                     [cplex.Model.A;E], ...
                                            [cplex.Model.rhs;zeros(n,1)], ...
                                            ...
251                                     cplex.Model.lb,cplex.Model.ub);
252                                 if exitflag ~= -2
253                                     GAb = Gtemp;
254                                     passcheck = 1;
255                                 end;
256                             end;
257                         end;
258                         tline = fgets(fin);
259                     end;
260                 end;
261                 gapIn = strcat(filePath,ProbName,'.gap.size');
262                 fsize = fopen(gapIn,'w');
263                 fprintf(fsize,strcat(GAb,'Size(GAb);\nquit;\n'));
264                 fclose(fsize);
265                 lin_com = strcat(gap,32,'<',32,gapIn);
266                 [~,grpsize] = system(lin_com);
267                 fprintf(fout,strcat(grpsize,','));
268                 fprintf(fout,'%d,',toc);
269
270                 SendMeResults(strcat(filePath,ProbName,'.csv'));
271                 fprintf(1,strcat(10,10,'Finishing',32, ...
                        ProbName,32,'double coset decomposition.',10,10));
272             else
273                 fprintf(fout,'G(A 0)^LP = G(A b)^LP');
274                 fprintf(1,strcat(10,10, 'G(A 0)^LP = G(A b)^LP ...
                        for',32,ProbName,32,10,10));
275             end;
276         else
277             fprintf(fout,',,,,,,,,');
```

```
278
279         end;
280     else
281         fprintf(fout,'GLP N/A,,,,,,,,');
282         fprintf(1,strcat(10,10, 'GLP does not apply ...
                for',32,ProbName,'.',10,10));
283     end;
284
285     fclose(fout);
286
287 end;
288 matlabpool('close');
289
290 fout = fopen('DoubleCosetResults.csv','w');
291 fprintf(fout,'Problem, super dfix, super numP, super grpsize, super ...
       time, sub dfix,sub numP, sub grpsize,sub time, G(A b)^LP grpsize, ...
       G(A b)^LP time\n');
292 for i = 1:numProb
293     ProbName = strrep(strrep(ProbList{i},'.gz',''),'.lp','');
294     filePath = strcat(ProbName,'/DoubleCoset/');
295     fileName = strcat(filePath,ProbName,'.csv');
296     fin = fopen(fileName,'r');
297     if fin > 1
298         fprintf(fout,strcat(ProbName,',',fgets(fin),'\n'));
299         fclose(fin);
300     else
301         fprintf(fout,strcat(ProbName,',FAILED\n'));
302     end;
303 end;
304
305 fclose(fout);
306
```

```
307 SendMeResults( 'DoubleCosetResults.csv' );
```

Appendix2/BuildProblem.m

```
 1
 2 function [ cplex, numslack ] = BuildProblem(fileName, addSlack, ...
       Aineq, bineq, Aeq, beq, c, lb, ub, ctype)
 3
 4 numslack = [];
 5
 6 % Add cplex to the MATLAB path
 7 addpath(genpath('/opt/ibm/ILOG/CPLEX_Studio1251/cplex/'));
 8
 9 % Aineq <= bineq
10
11 % The problem name and its file path
12 [filePath,ProbName,~] = fileparts(strrep(strrep(strrep(fileName, ...
13     '.mps',''),'.gz',''),'.lp',''));
14
15 if strcmp(filePath,pwd)
16     filePath = '';
17 end;
18
19 if ~isempty(filePath)
20     filePath = strcat(pwd,'/',strrep(filePath,strcat(pwd,'/'),''),'/');
21 end;
22
23 % Initialize cplex
24 cplex = Cplex();
25
26 % If the input matrices are empty and the fileName is an mps or lp ...
       file, use
27 % cplex to read in the mps file
```

```
28 if isempty(Aineq) && isempty(Aeq) && ...
       length(strrep(strrep(strrep(fileName, '.mps',''), '.lp',''), ...
       '.gz','')) < length(fileName)
29
30      % Read the model file into cplex
31      cplex.readModel(fileName);
32
33 elseif max(size(Aineq)) + max(size(Aeq)) == 0
34
35      fprintf(1,'You either need an mps or lp format input file or you ...
            need to enter the correct matrices.\n');
36      clear cplex;
37      return;
38
39 else
40
41      % Sense if it is min or max
42      cplex.Model.sense = 'minimize';
43
44      % The constraint matrix
45      cplex.Model.A = [Aeq;Aineq];
46
47      % The right hand side
48      cplex.Model.rhs = [beq;bineq];
49
50      % The left hand side
51      cplex.Model.lhs = [beq; -Inf*ones(size(bineq))];
52
53      % Objective function
54      cplex.Model.obj = c;
55
56      % Lower bounds
```

```
57      cplex.Model.lb = lb;
58
59      % Upper bounds
60      cplex.Model.ub = ub;
61
62      % If no variable type is given, we assume all are continuous
63      if isempty(ctype)
64          ctype = repmat('C',1, numel(c));
65      end;
66
67      % Types of variables
68      cplex.Model.ctype = ctype;
69
70  end;
71
72  % Find >= constraints
73  x = find(~isinf(cplex.Model.lhs') & cplex.Model.lhs' ~= ...
        cplex.Model.rhs');
74  if ~isempty(x)
75      % Find constraints that are both >= and <=
76      y = find(~isinf(cplex.Model.rhs(x)));
77      z = x(y);
78      clear y;
79      if ~isempty(z)
80          cplex.addRows(-Inf * ones(numel(z),1), -1 * ...
                cplex.Model.A(z,:), -1 * cplex.Model.lhs(z));
81      end;
82      clear z;
83
84      % Convert >= constraints to <=
85      y = find(isinf(cplex.Model.rhs(x)));
86      z = x(y);
```

```
87      clear y;
88      if ~isempty(z)
89          cplex.Model.rhs(z) = -1 * cplex.Model.lhs(z);
90          cplex.Model.A(z,:) = -1 * cplex.Model.A(z,:);
91          cplex.Model.lhs(z) = -Inf;
92      end;
93      clear z x;
94
95  end;
96
97  if addSlack == 1
98      ineqconst = find(isinf(cplex.Model.lhs'));
99      if ~isempty(ineqconst)
100         numDigit = SigDigit(cplex.Model.ub(~isinf(cplex.Model.ub)));
101         numslack = numel(ineqconst);
102         Z = zeros(size(cplex.Model.A,1), numslack);
103         ctype = '';
104         lb = zeros(numslack,1);
105         ub = Inf*ones(numslack,1);
106         for i = 1:1:numslack
107             Z(ineqconst(i),i) = 1;
108             ctype = strcat(ctype, 'C');
109         end;
110         cplex.addCols(zeros(numslack,1), Z, lb, ub, ctype);
111         cplex.Model.lhs(ineqconst) = cplex.Model.rhs(ineqconst);
112         numvar = size(cplex.Model.A, 2) - numslack;
113
114         x0 = [];
115         cplex.Model.ctype = strrep(cplex.Model.ctype, 'B','I');
116         for i = 1:1:numslack
117             y = [cplex.Model.A(ineqconst(i),:), ...
                    cplex.Model.rhs(ineqconst(i))];
```

```
118             findA = find(cplex.Model.A(ineqconst(i), 1:numvar) ~= 0);
119             ctype = cplex.Model.ctype(findA); %#ok<FNDSB>
120
121             if (isequal(floor(y),y) && length(ctype) == ...
                    length(strrep(ctype, 'C', '')))
122                 cplex.Model.ctype(numvar + i) = 'I';
123             end;
124
125             f = zeros(numvar+numslack, 1);  f(numvar+i) = -1;
126             [x0,x] = cplexlp(f, [], [], cplex.Model.A, ...
                    cplex.Model.rhs, cplex.Model.lb, cplex.Model.ub,x0);
127             if ~isempty(x)
128                 cplex.Model.ub(numvar+i) = -x;
129                 clear x;
130                 newub = find(cplex.Model.ctype(1:(numvar + i - 1)) ...
                        == cplex.Model.ctype(numvar + i) & ...
                        abs(cplex.Model.ub(1:(numvar + i - 1))' - ...
                        cplex.Model.ub(numvar+i)) <= 10^-6, 1);
131                 if ~isempty(newub)
132                     cplex.Model.ub(numvar+i) = cplex.Model.ub(newub);
133                 elseif cplex.Model.ctype(numvar + i) == 'I'
134                     if floor(cplex.Model.ub(numvar + i)) + 1 - ...
                            cplex.Model.ub(numvar + i) <= 10^-6
135                         cplex.Model.ub(numvar + i) = ...
                                floor(cplex.Model.ub(numvar + i)) + 1;
136                     else
137                         cplex.Model.ub(numvar + i) = ...
                                floor(cplex.Model.ub(numvar + i));
138                     end;
139                 else
140                     cplex.Model.ub(numvar + i) = ...
                            roundn(cplex.Model.ub(numvar + i), -numDigit);
```

```
141                 end;
142             end;
143
144         end;
145
146         findbinary = find(cplex.Model.lb' == 0 & cplex.Model.ub' == ...
                1 & cplex.Model.ctype == 'I');
147         if ~isempty(findbinary)
148             cplex.Model.ctype(findbinary) = 'B';
149         end;
150     end;
151 end;
152
153 if isempty(ProbName)
154     cplex.writeModel(strcat(filePath, 'bidon.lp'));
155 else
156     cplex.writeModel(strcat(filePath, ProbName,'.lp'));
157 end;
158
159 end
```

Appendix2/ProjMatrix.m

```
1 function [ P, numDigit ] = ProjMatrix( A, c, useAdj )
2 %
3
4 [m,n] = size(A);
5
6 %--------------Determine Projection or Adjacency ...
      Matrix--------------------
7
8 if useAdj == 0
9     % Append the constraint matrix to the equality constraints
```

```
10      A = [c';A];
11      % Find the projection matrix
12      [U,D,V] = svd(full(A));
13      numDigit = abs(floor(log10(max(max(abs(U*D*V' - A))))));
14      d = sum(sum(abs(D) >= 10^-9));
15      M = zeros(n);
16      M(1:1:d,1:1:d) = eye(d);
17      P = V*M*V';
18  else
19      numDigit = abs(floor(log10(eps*10)));
20      % Create graph adjacency matrix (Margot method)
21      P = [zeros(n) A'; A zeros(m)];
22  end;
23
24  %-------------------------------------------------------------------------
25
26
27  end
```

Appendix2/runHnosigneff.m

```
1  function [ dfix,g6fileName,numP ] = runHnosigneff(P, numDigit, ...
       filePath, ProbName)
2  % This is the MATLAB wrapper for Hnosigneff that returns dfix
3
4  Hnosigneff = '/scratch/Obj1/Hnosigneff';
5  PfileName = strcat(filePath, ProbName, '.P.txt');
6  g6fileName = strcat(filePath, ProbName, '.g6');
7
8  dfix = -1;
9
10 epsilon = 10^(-1*numDigit); % Numerical precision
11
```

```
12 numVert = size(P,1);
13
14 %----Save lower half of P as a vector in a text file-----
15 p = zeros(1,sum(1:numVert));
16 for i = 1:1:numVert
17     p(1,(sum(1:(i - 1)) + 1):(sum(1:(i - 1)) + i)) = P(i, 1:i);
18 end;
19 P = p;
20 clear p;
21 x = find(abs(P) <= epsilon);
22 if ~isempty(x)
23     P(x) = 0;
24 end;
25
26 fp = fopen(PfileName,'w');
27 for i = 1:1:numel(P)
28     if isequal(P(i),floor(P(i)))
29         fprintf(fp,'%i ',P(i));
30     else
31         fprintf(fp,strcat('%.', num2str(numDigit), 'f', 32), ...
               roundn(P(i), -1 - numDigit));
32     end;
33 end;
34 fclose(fp);
35
36 % Find unique entries that are more than 20 * epsilon apart
37 P = unique(dlmread(PfileName));
38 d = P(1);
39 for i = 2:1:numel(P);
40     if abs(d(numel(d))-P(i)) > 20 * epsilon
41         d = [d, P(i)]; %#ok<AGROW>
42     end;
```

```
43 end;
44 numP = max(size(d));
45
46 clear P d i;
47
48 %----Call Hnosigneff----
49 lin_com = strcat(Hnosigneff, 32, PfileName, 32, g6fileName, 32, ...
       num2str(numVert,'%i'), 32, num2str(20*epsilon, strcat('%.', ...
       num2str(numDigit + 1), 'f')));
50
51 [status, result] = system(lin_com);
52
53 if status ~= 0
54     AAout = strcat('Error from Hnosigneff:', 10, result, 10);
55     disp(AAout);
56     return;
57 end;
58
59  %---Load value for dfix----
60 j = 1;
61 while j < (length(result) - 1) && ~strcmp(result(j:(j + 1)), 'd=')
62     j = j + 1;
63 end;
64 j = j + 2;
65 q = j;
66 while q < length(result) && ~strcmp(result(q), ' ')
67     q = q + 1;
68 end;
69 dfix = str2double(strrep(result(j:q), ' ', ''));
70
71 if isnan(dfix)
72     dfix = -1;
```

```
73 end;
74
75 end
```

Appendix2/ColorGraph.m

```
1 function [ fcolor ] = ColorGraph(c, lhs, rhs, lb, ub, ctype, dfix, ...
      useAdj)
2 % This function makes the graph coloring text for nauty
3
4 n = numel(c);
5 x = 1:1:n;
6
7 % --- Graph coloring for variables ---
8 if useAdj == 0
9 % If GLP, we do not care about the objective function coefficient
10   colorlist = [lb, ub, abs(ctype)'];
11 else
12   colorlist = [lb, ub, abs(ctype)', c];
13 end;
14
15 fcolor = '\nf=[';
16
17 while ~isempty(x)
18   y = find(ismember(colorlist, colorlist(x(1), :), 'rows')==1)';
19   for k = 1:1:numel(y)
20     fcolor = strcat(fcolor, num2str((dfix * (y(k) - 1)):1:(dfix * ...
          y(k) - 1), '%i, '));
21   end;
22   fcolor = strcat(fcolor, '|');
23   x = setdiff(x, y);
24 end;
25
```

```
26 % --- Graph coloring for vertices ---
27 if useAdj == 1
28   m = numel(rhs);
29   x = 1:1:m;
30   colorlist = [lhs, rhs];
31   while ~isempty(x)
32     y = find(ismember(colorlist, colorlist(x(1), :), 'rows')==1)';
33     for k = 1:1:numel(y)
34       fcolor = strcat(fcolor, num2str(((dfix * (y(k) - 1)):1:(dfix * ...
             y(k) - 1)) + (dfix * n), '%i, '));
35     end;
36     fcolor = strcat(fcolor, '|');
37     x = setdiff(x, y);
38   end;
39
40 end;
41
42 fcolor = strrep(strrep(strrep(strcat(fcolor, ']'), ', |]', ']'), ', ...
     |', ' | ') , ', ', ', ');
43
44 return
45
46 % ...
     ------------------------------------------------------------------------
47
48 if useAdj == 0
49
50   colorlist = [lb(1), ub(1), abs(ctype(1))];
51   varcolors = zeros(n, 1);
52   varcolors(1) = 1;
53   for i = 2:1:n
54     j = 1;
```

```
55      while j <= size(colorlist, 1) && ~isequal(colorlist(j, :), ...
            [lb(i), ub(i), abs(ctype(i))])
56        j = j + 1;
57      end;
58      if j > size(colorlist, 1)
59        colorlist = [colorlist; lb(i), ub(i), abs(ctype(i))]; %#ok<AGROW>
60      end;
61      varcolors(i) = j;
62    end;
63  elseif useAdj == 1
64    colorlist = [c(1), lb(1), ub(1), abs(ctype(1))];
65    varcolors = zeros(n, 1);
66    varcolors(1) = 1;
67    for i = 2:1:n
68      j = 1;
69      while j <= size(colorlist, 1) && ~isequal(colorlist(j, :), ...
            [c(i), lb(i), ub(i), abs(ctype(i))])
70        j = j + 1;
71      end;
72      if j > size(colorlist, 1)
73        colorlist = [colorlist; c(i), lb(i), ub(i), abs(ctype(i))]; ...
              %#ok<AGROW>
74      end;
75      varcolors(i) = j;
76    end;
77  end;
78
79  clear colorlist;
80
81  fcolor = '';
82
83  maxcol = max(varcolors);
```

```
84 if maxcol > 1
85   fcolor = '\nf=[';
86   for i = 1:1:maxcol
87     j = find(varcolors==i);
88     q = max(size(j));
89     for k = 1:1:q
90       fcolor = strcat(fcolor, num2str((dfix * (j(k) - 1)):1:(dfix * ...
             j(k) - 1), '%i, '));
91     end;
92     fcolor = strcat(fcolor, '|');
93   end;
94   fcolor = strrep(strrep(strrep(strcat(fcolor, ']'), ', |]', ']'), ...
       ', |', ' | ') , ', ', ', ');
95 end;
96
97 clear varcolors;
98
99 if useAdj == 1
100
101 % --- Graph coloring for vertices ---
102
103   b = [beq;bineq];
104   contype = 0;
105   if m == 0
106     contype = 1;
107   end;
108   colorlist = [b(1), contype];
109   concolors = zeros(m + p, 1);
110   concolors(1) = 1;
111   for i=2:1:(m + p)
112     if m < i
113       contype = 1;
```

```
114     end;
115     j = 1;
116     while j <= size(colorlist, 1) && ~isequal(colorlist(j, :), ...
            [b(i), contype])
117       j = j + 1;
118     end;
119     if j > size(colorlist, 1)
120       colorlist = [colorlist; b(i), contype]; %#ok<AGROW>
121     end;
122     concolors(i) = j;
123   end;
124
125   clear colorlist b;
126
127   maxconcol = max(concolors);
128   if maxconcol > 1
129     if maxcol > 1
130       fcolor = strrep(fcolor, ']', ', |');
131     else
132       fcolor = '\nf=[';
133     end;
134     for i = 1:1:maxconcol
135       j = find(concolors==i);
136       q = max(size(j));
137       for k = 1:1:q
138         fcolor = strcat(fcolor, num2str(((dfix * (j(k) - 1)):1:(dfix ...
                * j(k) - 1)) + (dfix * n), '%i, '));
139       end;
140       fcolor = strcat(fcolor, '|');
141     end;
142     fcolor = strrep(strrep(strrep(strcat(fcolor, ']'), ', |]', ']'), ...
          ', |', ' | ') , ', ', ', ');
```

```
143   end;
144
145   clear maxcol maxconcol concolors;
146
147 end;
148
149 end
```

Appendix2/NautyGroup.m

```
1 function [ grpsize, dreadOutPerm ] = NautyGroup( ...
      g6fileName,fcolor,Cycle)
2 % Uses Nauty to determine group size for a given g6 format graph ...
      colored in accordance with fcolor.
3
4 grpsize = '';
5
6 if Cycle <= 0
7     perm = '\np';
8 else
9     perm = '';
10 end;
11
12 [filePath,ProbName,~] = fileparts(g6fileName);
13
14 if strcmp(filePath,pwd)
15     filePath = '';
16 end;
17
18 if ~isempty(filePath)
19     filePath = strcat(pwd, '/', strrep(filePath, strcat(pwd,'/'), ...
          ''), '/');
20 end;
```

```
21
22 junk = strcat(filePath, ProbName,'.junk');
23 dreadInPerm = strcat(filePath, ProbName,'.naut.inp');
24 dreadOutPerm = strcat(filePath, ProbName,'.aa.out');
25
26 %----Nauty Linux command line scripts----
27 [~,hostname] = system('hostname');
28 if strcmp(cellstr(hostname),'Deep-Thought')
29     nautyFile = '/etc/nauty25r5/';                % Folder ...
          containing nauty
30 else
31     nautyFile = '/usr/local/nauty25/';            % Folder ...
          containing nauty
32 end;
33 listg = strcat(nautyFile, 'listg -d');             % listg executable
34 dreadnaut = strcat(nautyFile, 'dreadnaut <');      % dreadnaut ...
      executable
35
36 %----Call listg----
37 lin_com = strcat(listg, 32, g6fileName, 32, '>', 32, junk);
38
39 [status, ~] = system(lin_com);
40 if status ~= 0
41     AAout = strcat('Error from listg:', 10, result, 10);
42     disp(AAout);
43     return;
44 end;
45
46 %---Make input file for dreadnaut (permutations)----
47 fperm = fopen(dreadInPerm, 'w');
48 fprintf(fperm,strcat('<', 32, junk, fcolor, perm,'\n?\nx\n'));
49 fclose(fperm);
```

```
50
51 clear fcolor;
52
53 %---Call dreadnaut (cycles)----
54 lin_com = strcat(dreadnaut, 32, dreadInPerm);
55
56 [status, AAout] = system(lin_com);
57
58 %---Get rid of 'Mode=dense' in nauty output so it can be read by ...
       Margot---
59 AAout = strrep(AAout,'Mode=dense ', '');
60
61 if status ~= 0
62     AAout = strcat('Error from dreadnaut:', 10, AAout, 10);
63     disp(AAout);
64     return;
65 end;
66
67 %---Make output file for dreadnaut (cycles)----
68 fout = fopen(dreadOutPerm, 'w');
69 fprintf(fout,AAout);
70 fclose(fout);
71
72  %---Load value for groupsize----
73 j = 1;
74 while j < (length(AAout)-1) && ...
75         ~strcmp(AAout(j:(j + 7)),'grpsize=')
76     j = j + 1;
77 end;
78 j = j + 8;
79 q = j;
80 while q < length(AAout) && ~strcmp(AAout(q + 1), ';')
```

```
81      q = q + 1;
82  end;
83  grpsize = AAout(j:q);
84
85  if isnan(str2double(grpsize))
86      grpsize = '';
87  end;
88
89  end
```

Appendix2/OrbitProj.m

```
1  function E = OrbitProj(filePath,ProbName,numvar,G, GrpName )
2
3  [~,hostname] = system('hostname');
4
5  %----Call Gap command----
6  if strcmp(cellstr(hostname),'Deep-Thought')
7      gap = '/etc/gap4r6/bin/gap-default64.sh -m 20000m -q';
8  elseif strcmp(cellstr(hostname),'ensphd01')
9      gap = '/usr/local/gap4r5/gap.shi -m 20000m -q';
10 else
11     gap = 'gap.sh -m 20000m -q';
12 end;
13
14 ProbName = strrep(ProbName,'.aa','');
15 gapIn = strcat(filePath,ProbName,'.gap.inp');
16 gapOut = strcat(filePath,ProbName,'.gap.orb');
17
18 %-----------------------------------------
19
20 %-----------Gap command file-----------------
21 fout = fopen(gapIn,'w');
```

```
22 fprintf(fout,G);
23 fprintf(fout,strcat('Orbits(', GrpName, ',[1..', num2str(numvar), ...
       ']);\nquit;'));
24 fclose(fout);
25 E = [];
26
27 if length(strrep(G,'Group(())',''')) == length(G)
28
29     lin_com = strcat(gap,32,'<',32,gapIn);
30     [~, result] = system(lin_com);
31     result = strrep(result,char(32),'');
32     result = strrep(result,'[','');
33     result = strrep(result,']]','');
34     result = strrep(result,char(10),'');
35     result = strrep(result,'],',char(10));
36     fout = fopen(gapOut,'w');
37         fprintf(fout,result); fprintf(fout,'\n');
38     fclose(fout);
39     fin = fopen(gapOut,'r');
40     tline = fgets(fin);
41     while ischar(tline)
42         x = str2num(tline); %#ok<ST2NM>
43         if ~isempty(x)
44             Y = zeros(numvar,1);
45             Y(x) = 1;
46             E = [E, Y]; %#ok<AGROW>
47         end;
48         tline = fgets(fin);
49     end;
50     [~,D,V] = svd(E');
51     d = sum(sum(abs(D) >= 10^-9));
52     M = zeros(numvar);
```

```
53      M(1:1:d,1:1:d) = eye(d);
54      E = eye(numvar)-V*M*V';
55  end;
56
57  end
```

Appendix2/Nauty2Gap.m

```
 1  function GroupString = Nauty2Gap(dfix, numvar, infile, GroupName)
 2  % This function removes layers for the GLP group
 3
 4  fin = fopen(infile);
 5
 6  tline = fgets(fin);
 7  cyclestring = '';
 8  redstring = '';
 9  GroupString = strcat(GroupName,':=Group(');
10
11  while ischar(tline)
12
13      if tline(1) == '('
14          if ~isempty(cyclestring)
15              X = [find(cyclestring ==')' + 1, ...
                    find(cyclestring==')')' - 1];
16              numcycles = size(X,1);
17              for i = 1:1:numcycles
18                  x = str2num(cyclestring(X(i,1):X(i,2))); %#ok<ST2NM>
19                  x = floor(x./dfix) + 1;
20                  j = 2;
21                  while j <= size(x,2)
22                      if ismember(x(j),x(1:j-1))
23                          x(j) = [];
24                      else
```

```
25                          j = j + 1;
26                      end;
27                  end;
28                  if ~isempty(x) && max(x) <= numvar
29                      xstring = strrep(strcat('(',num2str(x, ...
                            '%d,'),')'),',)',')');
30                      if length(strrep(redstring,xstring,'')) == ...
                            length(redstring)
31                          redstring = strcat(redstring, xstring);
32                      end;
33                  end;
34              end;
35              GroupString = strcat(GroupString, redstring,',');
36              redstring = '';
37          end;
38          cyclestring = strtrim(tline);
39      elseif strcmp(tline(1:3),'   ')
40          cyclestring = strcat(strtrim(cyclestring), ',', strtrim(tline));
41      elseif ~isempty(cyclestring)
42          X = [find(cyclestring =='(')' + 1, find(cyclestring==')')' - 1];
43          numcycles = size(X,1);
44          for i = 1:1:numcycles
45              x = str2num(cyclestring(X(i,1):X(i,2))); %#ok<ST2NM>
46              x = floor(x./dfix) + 1;
47              j = 2;
48              while j <= numel(x)
49                  if ismember(x(j),x(1:j-1))
50                      x(j) = [];
51                  else
52                      j = j + 1;
53                  end;
54              end;
```

```
55              if ~isempty(x) && max(x) <= numvar
56                  xstring = strrep(strcat('(', num2str(x,'%d,'), ')'), ...
                        ',)', ')');
57                  if length(strrep(redstring, xstring, '')) == ...
                        length(redstring)
58                      redstring = strcat(redstring, xstring);
59                  end;
60              end;
61          end;
62          if ~isempty(redstring)
63              GroupString = strcat(GroupString, redstring,',');
64          end;
65          cyclestring = '';
66          redstring = '';
67      end;
68      tline = fgets(fin);
69
70  end;
71
72  fclose(fin);
73  GroupString = GroupString(1:max(9, length(GroupString) - 1));
74  GroupString = strrep(strcat(GroupString, ');;\n'), 'Group();;', ...
        'Group(());;');
75
76  end
```

Appendix2/DoubleCoset.m

```
1  function [gapOut] = DoubleCoset(filePath,ProbName,G, Gname, M, Mname )
2
3  [~,hostname] = system('hostname');
4
5  %----Call Gap command----
```

```
6 if strcmp(cellstr(hostname),'Deep-Thought')
7     gap = '/etc/gap4r6/bin/gap-default64.sh -m 20000m -q';
8 elseif strcmp(cellstr(hostname),'ensphd01')
9     gap = '/usr/local/gap4r5/gap.shi -m 20000m -q';
10 else
11     gap = 'gap.sh -m 20000m -q';
12 end;
13
14 ProbName = strrep(ProbName,'.aa','');
15 gapIn = strcat(filePath,ProbName,'.gap.inp');
16 gapOut = strcat(filePath,ProbName,'.gap.rep');
17
18 %---------Gap command file-------------
19 fout = fopen(gapIn,'w');
20 fprintf(fout,M);
21 fprintf(fout,G);
22 fprintf(fout,strcat('dc:=DoubleCosets(', Gname,',', Mname,',', ...
      Mname, ');;\nList(dc,Representative);\nquit;'));
23 fclose(fout);
24
25 if length(strrep(M,'Group(())','')) == length(M) && ...
      length(strrep(G,'Group(())','')) == length(G)
26
27     lin_com = strcat(gap,32,'<',32,gapIn);
28     [~, result] = system(lin_com);
29     result = strrep(strrep(strrep(strrep(strrep(result, ...
          '(),', '') ,']', ''), '[', ''), char(10),''), char(32), ''), ...
          '),' ,')\n');
30     fout = fopen(gapOut,'w');
31         fprintf(fout,result); fprintf(fout,'\n');
32     fclose(fout);
33
```

```
34 end;
35
36 end
```

Appendix2/SendMeResults.m

```
1 function SendMeResults( attachment )
2 % This emails me the file
3
4 mySMTP = 'ms-afit-03.afit.edu';
5 myEmail = 'andrew.geyer@afit.edu';
6 setpref('Internet', 'SMTP_Server', mySMTP);
7 setpref('Internet', 'E_mail', myEmail);
8
9 %---------------------------------
10
11 sendmail(myEmail, 'code results', strcat('Hey,', 10, 10, 'Here are ...
      the results.', 10, 10, 'You'), attachment);
12
13 end
```

Bibliography

[1] Appa, G., D. Magos, and I. Mourtos. "On Multi-Index Assignment Polytopes". *Linear Algebra and its Applications*, 416(23):224 – 241, 2006. ISSN 0024-3795. URL http://www.sciencedirect.com/science/article/pii/S002437950500563X.

[2] Appa, G., I. Mourtos, and D. Magos. "A Branch & Cut Algorithm for a Four-Index Assignment Problem". *Journal of the Operational Research Society*, 55(3):298–307, 2004. URL http://dx.doi.org/10.1057/palgrave.jors.2601655.

[3] Appa, G., I. Mourtos, and D. Magos. "Integrating Constraint and Integer Programming for the Orthogonal Latin Squares Problem". P. Hentenryck (editor), *Principles and Practice of Constraint Programming - CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, 17–32. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-44120-5. URL http://dx.doi.org/10.1007/3-540-46135-3_2.

[4] Bödi, R., K. Herr, and M. Joswig. "Algorithms for Highly Symmetric Linear and Integer Programs". *Mathematical Programming*, 137(1-2):65–90, 2013. ISSN 0025-5610. URL http://dx.doi.org/10.1007/s10107-011-0487-6.

[5] Bremner, D., M.D. Sikiric, D.V. Pasechnik, and A. Schuermann. "Computing Symmetry Groups of Polyhedra". *arXiv preprint arXiv:1210.0206*, 2013.

[6] Bulutoglu, D.A. Personal Communication, October 2013.

[7] Bulutoglu, D.A. and F. Margot. "Classification of Orthogonal Arrays by Integer Programming". *Journal of Statistical Planning and Inference*, 138(3):654–666, 2008.

[8] Bulutoglu, D.A. and K.J. Ryan. "Integer Programming for Finding Generalized Minimum Aberration Designs". *Submitted to Operations Research*, 2013.

[9] Espinoza, D.G. "On Linear Programming, Integer Programming and Cutting Planes". 2006.

[10] Fecko, M.A. and M. Steinder. "Combinatorial Designs in Multiple Faults Localization for Battlefield Networks". *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, volume 2, 938–942. IEEE, 2001.

[11] Fripertinger, H. "Enumeration of the Semilinear Isometry Classes of Linear Codes". *Bayreuther Mathematische Schriften*, (74):100–122, 2005. ISSN 0172-1062.

[12] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.6.4*, 2013. URL http://www.gap-system.org.

[13] Hedayat, A, N.J.A. Sloane, and J. Stufken. *Orthogonal Arrays: Theory and Applications*. Springer Verlag, 1999.

[14] Hill, R. and S. Chambal. "A Framework for Improving Experimental Design and Statistical Methods for Test and Evaluation". *AIAA T&E Days*, 2009.

[15] Holzmann, W.H., H. Kharaghani, and B. Tayfeh-Rezaie. "Williamson Matrices Up to Order 59". *Designs, Codes and Cryptography*, 46(3):343–352, 2008. ISSN 0925-1022. URL http://dx.doi.org/10.1007/s10623-007-9163-5.

[16] Hutto, G.T. and J.M. Higdon. "Survey of Design of Experiments (DOE) Projects in Developmental Test CY07-08". *American Institute of Aeronautics and Astronautics 2009*, 1706, 2009.

[17] Leon, S.J. *Linear Algebra With Applications*. Maxwell Macmillan international editions. MacMillan, 1990. ISBN 9780023698217.

[18] Liberti, L. "Reformulations in Mathematical Programming: Automatic Symmetry Detection and Exploitation". *Mathematical Programming*, 131(1-2):273–304, 2012.

[19] Margot, F. "Pruning by Isomorphism in Branch-and-Cut". *Mathematical Programming*, 94(1):71–90, 2002.

[20] Margot, F. "Exploiting Orbits in Symmetric ILP". *Mathematical Programming*, 98(1-3):3–21, 2003.

[21] Margot, F. "Small Covering Designs by Branch-and-Cut". *Mathematical Programming*, 94(2-3):207–220, 2003.

[22] Margot, F. "Symmetric ILP: Coloring and Small Integers". *Discrete Optimization*, 4(1):40–62, 2007.

[23] Margot, F. "Symmetry in Integer Linear Programming". *50 Years of Integer Programming 1958-2008*, 647–686. Springer, 2010.

[24] McKay, B.D. "Hadamard Equivalence Via Graph Isomorphism". *Discrete Mathematics*, 27(2):213–214, 1979. ISSN 0012-365X. URL http://www.sciencedirect.com/science/article/pii/0012365X79901134.

[25] McKay, B.D. "Nauty Users Guide (version 2.5)". *Computer Science Dept., Australian National University*, 2013.

[26] Montgomery, D.C. *Design and Analysis of Experiments*. Design and Analysis of Experiments. Wiley, 2012. ISBN 9781118146927.

[27] Ostrowski, J., J. Linderoth, F. Rossi, and S. Smriglio. "Constraint Orbital Branching". *Integer Programming and Combinatorial Optimization*, 225–239. Springer, 2008.

[28] Panetta, L.E. and B.H. Obama. *Sustaining US Global Leadership: Priorities for 21st Century Defense*. Department of Defense, 2012.

[29] Rosenberg, S.J. "A Large Index Theorem for Orthogonal Arrays, With Bounds". *Discrete Mathematics*, 137(1):315–318, 1995.

[30] Salvagnin, D. "A Dominance Procedure for Integer Programming". *Master's Thesis, University of Padua*, 2005.

[31] Seberry, J. and M. Yamada. "Hadamard Matrices, Sequences, and Block Designs". 1992.

[32] Stufken, J. and B. Tang. "Complete Enumeration of Two-Level Orthogonal Arrays of Strength d With d+2 Constraints". *The Annals of Statistics*, 35(2):793–814, 04 2007. URL http://dx.doi.org/10.1214/009053606000001325.

[33] Tucker, A.A., G.T. Hutto, and C.H. Dagli. "Application of Design of Experiments to Flight Test: A Case Study". *United States Air Force*, 2008.

[34] Williamson, J. "Hadamards Determinant Theorem and the Sum of Four Squares". *Duke Mathematical Journal*, 11(1):65–81, 1944.

[35] Xu, H and C.F.J. Wu. "Generalized Minimum Aberration for Asymmetrical Fractional Factorial Designs". *The Annals of Statistics*, 29(2):549–560, 2001.

Vita

Major Andrew J. Geyer graduated from Minot High School in Minot, North Dakota in May 1996. He entered undergraduate studies at North Dakota State University in Fargo, North Dakota where he graduated with a Bachelor of Science degree in Physics in May 2000. He was commissioned through Detachment 610, AFROTC at North Dakota State University.

His first assignment was to the Air Force Institute of Technologys Basic Meteorology Program at Texas A&M University in College Station, Texas. In September 2001, Maj Geyer was assigned to the 18th Weather Squadron at Fort Bragg, NC, where he served combat tours in Operations Enduring Freedom and Iraqi Freedom as a Combat Weather Team leader attached to the U.S. Army's 82nd Airborne Division. In May 2004, Maj Geyer reported to Kunsan Air Base, Republic of Korea, where he served as the weather flight commander for the 8th Fighter Wing. In July 2005, he reported to Fort Benning, Georgia where he served as Commander, Detachment 4, 10th Combat Weather Squadron and as the Staff Weather Officer for the U.S. Army's 75th Ranger Regiment. During that assignment, Maj Geyer deployed as the Joint METOC Officer assigned to Combined Joint Special Operations Task Force-Arabian Peninsula in Iraq. In August 2007, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology where he received a Master of Science in Operations Research. In 2009, he was assigned to Headquarters, Air Force Weather Agency at Offutt Air Force Base, Nebraska. In 2010, he served as the Senior Meteorology Officer for the International Security Assistance Force in Kabul, Afghanistan. In 2011, Maj Geyer returned to the Graduate School of Engineering and Management, Air Force Institute of Technology where he pursued a Doctor of Philosophy in Applied Mathematics - Statistics.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
19–06–2014	Doctoral Dissertation	Oct 2011–Jun 2014

4. TITLE AND SUBTITLE

Different Formulations of the Orthogonal Array Problem and Their Symmetries

5a. CONTRACT NUMBER

5b. GRANT NUMBER

F1ATA03039J001

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Geyer, Andrew J., Major, USAF

5d. PROJECT NUMBER

14C239T

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENC-DS-14-J-16

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research
Dr. Fariba Fahroo AFOSR/RTA
DSN 426-8429
Opt.DMath@afosr.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A: Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Modern statistical experiments routinely feature a large number of input variables that can each be set to a variety of different levels. In these experiments, output response changes as a result of changes in the individual factor level settings. Often, an individual experimental run can be costly in time, money or both. Therefore, experimenters generally want to gain the desired information on factor effects from the smallest possible number of experimental runs. Orthogonal arrays provide the most desirable designs. However, finding orthogonal arrays is a very challenging problem. There are numerous integer linear programming formulations (ILP) in the literature whose solutions are orthogonal arrays. Because of the nature of orthogonal arrays, these ILP formulations contain symmetries where some portion of the variables in the formulation can be swapped without changing the ILP. These symmetries make it possible to eliminate large numbers of infeasible or equivalent solutions quickly, thereby greatly reducing the time required to find all non-equivalent solutions to the ILPs.
In this dissertation, a new method for identifying symmetries is developed and tested using several existing and new ILP formulations for enumerating orthogonal arrays.

15. SUBJECT TERMS

Factorial Designs, Integer Programming, Orthogonal Arrays, Symmetry Group

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Dursun A. Bulutoglu (ENC)
U	U	U	UU	169	19b. TELEPHONE NUMBER *(include area code)* (937) 255-6565 x4704 dursun.bulutoglu@afit.edu